
Faucet Documentation

Faucet Developers

Jun 05, 2018

Contents

1	User Documentation	1
1.1	Introduction to Faucet	1
1.2	Tutorials	4
1.3	Installation	38
1.4	Docker	43
1.5	Configuration	45
1.6	Configuration Recipe Book	61
1.7	Vendor-specific Documentation	61
1.8	External Resources	85
2	Developer Documentation	87
2.1	Developer Guide	87
2.2	Architecture	90
2.3	Testing	94
2.4	Fuzzing	97
2.5	Source Code	98
3	Quick References	147
3.1	Frequently Asked Questions	147
4	Indices and tables	151
	Python Module Index	153

1.1 Introduction to Faucet

1.1.1 What is Faucet?

Faucet is a compact open source OpenFlow controller, which enables network operators to run their networks the same way they do server clusters. Faucet moves network control functions (like routing protocols, neighbor discovery, and switching algorithms) to vendor independent server-based software, versus traditional router or switch embedded firmware, where those functions are easy to manage, test, and extend with modern systems management best practices and tools. Faucet controls OpenFlow 1.3 hardware which delivers high forwarding performance.

You can read more about our approach to networking by reading our ACM Queue article [Faucet: Deploying SDN in the Enterprise](#).

1.1.2 What is Gauge?

Faucet has two main OpenFlow controller components, Faucet itself, and Gauge. Faucet controls all forwarding and switch state, and exposes its internal state, e.g. learned hosts, via Prometheus (so that an open source NMS such as Grafana graph it).

Gauge also has an OpenFlow connection to the switch and monitors port and flow state (exporting it to Prometheus or InfluxDB, or even flat text log files). Gauge, however, does not ever modify the switch's state, so that switch monitoring functions can be upgraded, restarted, without impacting forwarding.

1.1.3 Why Faucet?

Design

Faucet is designed to be very small, simple (1000s of lines of code, versus millions in other systems), and keep relatively little state. Faucet does not have any implementation-specific or vendor driver code, which considerably reduces complexity. Faucet does not need connectivity to external databases for forwarding decisions. Faucet provides

“hot/hot” high availability and scales through the provisioning of multiple Faucets with the same configuration - Faucet controllers are not inter-dependent.

Performance and scaling

As well as being compact, Faucet offloads all forwarding to the OpenFlow switch, including flooding if emulating a traditional switch. Faucet programs the switch pre-emptively, though will receive packet headers from the switch if, for example, a host moves ports so that the switch’s OpenFlow FIB can be updated (again, if traditional switching is being emulated). In production, Faucet controllers have been observed to go many seconds without needing to process a packet from a switch. In cold start scenarios, Faucet has been observed to completely program a switch and learn connected hosts within a few seconds.

Faucet uses a multi-table packet processing pipeline as shown in [Faucet Openflow Switch Pipeline](#). Using multiple flow tables over a single table allows Faucet to implement more complicated flow-based logic while maintaining a smaller number of total flows. Using dedicated flow tables with a narrow number of match fields, or limiting a table to exact match only, such as the IPv4 or IPv6 FIB tables allows us to achieve greater scalability over the number of flow entries we can install on a datapath.

A large network with many devices would run many Faucets, which can be spread over as many (or as few) machines as required. This approach scales well because each Faucet uses relatively few server resources and Faucet controllers do not have to be centralized - they can deploy as discrete switching or routing functional units, incrementally replacing (for example) non-SDN switches or routers.

An operator might have a controller for an entire rack, or just a few switches, which also reduces control plane complexity and latency by keeping control functions simple and local.

Testing

Faucet follows open source software engineering best practices, including unit and systems testing (python unittest based), as well static analysis (pytype, pylint, and codecov) and fuzzing (python-af). Faucet’s systems tests test all Faucet features, from switching algorithms to routing, on virtual topologies. However, Faucet’s systems tests can also be configured to run the same feature tests on real OpenFlow hardware. Faucet developers also host regular PlugFest events specifically to keep switch implementations broadly synchronized in capabilities and compatibility.

1.1.4 Release Notes

1.7.0

We are making a few potentially breaking features in faucet 1.7.0. This document covers how to navigate the changes and safely upgrade from earlier versions to 1.7.0.

1. Configuration and log directory changed

Starting in 1.7.0 and onwards faucet has changed which directories it uses for configuration and log files. The new paths are:

Old path	New path
/etc/ryu/faucet	/etc/faucet
/var/log/ryu/faucet	/var/log/faucet

Faucet 1.7.0 when being installed by pip will automatically attempt to migrate your old configuration files to `/etc/faucet` assuming it has permissions to do so. Failing this faucet when started will fallback to loading configuration from `/etc/ryu/faucet`. The search paths for configuration files are documented on the [Environment variables](#) page.

Note: Consider the `/etc/ryu/faucet` directory deprecated, we will in a future release stop reading config files stored in this directory.

If you currently set your own configuration or log directory by setting the appropriate environment variables you will be unaffected. In most other cases the migration code or the fallback configuration search path will allow the upgrade to 1.7.0 to be seamless. We have however identified two cases where manual intervention is required:

Dockers

Dockers will need to be started with new mount directories, the commands to start a 1.7.0 docker version of faucet or gauge are detailed in the [Docker](#) section.

Virtualenvs

We are unable to migrate configuration files automatically when faucet is run inside of a virtualenv, please copy the configuration directory over manually.

2. Changing default flood mode

Currently faucet defaults to using `combinatorial_port_flood` when it comes to provisioning flooding flows on a datapath, faucet implicitly configures a datapath like this today:

```
dps:
  mydp:
    combinatorial_port_flood: True
```

The default is `True`, in 1.7.0 and previously. The default will change to `False` in 1.7.1.

When `True`, flood rules are explicitly generated for each input port, to accommodate early switch implementations which (differing from the OpenFlow standard - see below) did not discard packets output to the packet input port. `False` generates rules per faucet VLAN which results in fewer rules and better scalability.

See [OpenFlow 1.3.5 specification](#), section B.6.3:

```
The behavior of sending out the incoming port was not clearly defined
in earlier versions of the specification. It is now forbidden unless
the output port is explicitly set to OFPP_IN_PORT virtual port
(0xffff8) is set.
```

1.1.5 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 Tutorials

1.2.1 Installing faucet for the first time

This tutorial will run you through the steps of installing a complete faucet system for the first time.

We will be installing and configuring the following components:

Component	Purpose
faucet	Network controller
gauge	Monitoring controller
prometheus	Monitoring system & time series database
grafana	Monitoring dashboard

This tutorial was written for Ubuntu 16.04, however the steps should work fine on any newer supported version of Ubuntu or Debian.

Package installation

1. Add the faucet official repo to our system:

```
sudo apt-get install curl gnupg apt-transport-https lsb-release
echo "deb https://packagecloud.io/faucetsdn/faucet/${lsb_release -si | awk '
↳ {print tolower($0)}')/ ${lsb_release -sc} main" | sudo tee /etc/apt/sources.
↳ list.d/faucet.list
curl -L https://packagecloud.io/faucetsdn/faucet/gpgkey | sudo apt-key add -
sudo apt-get update
```

2. Install the required packages, we can use the `faucet-all-in-one` metapackage which will install all the correct dependencies.

```
sudo apt-get install faucet-all-in-one
```

Configure prometheus

We need to configure prometheus to tell it how to scrape metrics from both the faucet and gauge controllers. To help make life easier faucet ships a sample configuration file for prometheus which sets it up to scrape a single faucet and gauge controller running on the same machine as prometheus. The configuration file we ship looks like:

Listing 1: prometheus.yml

```
# my global config
global:
  scrape_interval:      15s # Set the scrape interval to every 15 seconds. Default is
↳ every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
↳ minute.
  # scrape_timeout is set to the global default (10s).

# Load rules once and periodically evaluate them according to the global 'evaluation_
↳ interval'.
rule_files:
  - "faucet.rules.yml"
```

(continues on next page)

(continued from previous page)

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from
  ↳ this config.
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'faucet'
    static_configs:
      - targets: ['localhost:9302']
  - job_name: 'gauge'
    static_configs:
      - targets: ['localhost:9303']
```

To learn more about what this configuration file does you can look at the [Prometheus Configuration Documentation](#). The simple explanation is that it includes an additional `faucet.rules.yml` file that performs some automatic queries in prometheus for generating some additional metrics as well as setting up scrape jobs every 15 seconds for faucet listening on `localhost:9302` and gauge listening on `localhost:9303`.

Steps to make prometheus use the configuration file shipped with faucet:

1. Change the configuration file prometheus loads by editing the file `/etc/default/prometheus` to look like:

Listing 2: `/etc/default/prometheus`

```
# Set the command-line arguments to pass to the server.
ARGS="--config.file=/etc/faucet/prometheus/prometheus.yml"
```

2. Restart prometheus to apply the changes:

```
sudo systemctl restart prometheus
```

Configure grafana

Grafana running in it's default configuration will work just fine for our needs. We will however need to make it start on boot, configure prometheus as a data source and add our first dashboard:

1. Make grafana start on boot and then start it manually for the first time:

```
sudo systemctl daemon-reload
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

2. To finish setup we will configure grafana via the web interface.

First load `http://localhost:3000` in your web browser (by default both the username and password are admin).

3. The web interface will first prompt us to add a data source. Use the following settings then click Save & Test:

```
Name:    Prometheus
Type:    Prometheus
URL:     http://localhost:9090
```

4. Next we want to add some dashboards so that we can later view the metrics from faucet.

Hover over the + button on the left sidebar in the web interface and click `Import`.

We will import the following dashboards, just download the following links and upload them through the grafana dashboard import screen:

- [Instrumentation](#)
- [Inventory](#)
- [Port Statistics](#)

Configure faucet

For this tutorial we will configure a very simple network topology consisting of a single switch with two ports.

1. Configure faucet

We need to tell faucet about our topology and VLAN information, we can do this by editing the faucet configuration `/etc/faucet/faucet.yaml` to look like:

Listing 3: `/etc/faucet/faucet.yaml`

```
vlan:
  office:
    vid: 100
    description: "office network"

dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: office
      2:
        name: "host2"
        description: "host2 network namespace"
        native_vlan: office
```

Note: Tabs are forbidden in the YAML language, please use only spaces for indentation.

This will create a single VLAN and a single datapath with two ports.

2. Verify configuration

The `check_faucet_config` command can be used to verify faucet has correctly interpreted your configuration before loading it. This can avoid shooting yourself in the foot by applying configuration with typos. We recommend either running this command by hand or with automation each time before loading configuration.

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

This script will either return an error, or in the case of successfully parsing the configuration it will return a JSON object containing the entire faucet configuration that would be loaded (including any default settings), for example:

```
{'drop_spoofed_faucet_mac': True, 'hardware': 'Open vSwitch', 'lowest_priority': 0, 'highest_priority': 9099, 'faucet_dp_mac': '0e:00:00:00:00:01', 'metrics_rate_limit_sec': 0, 'use_idle_timeout': False, 'max_resolve_backoff_time': 32, 'high_priority': 9001, 'timeout': 300, 'pipeline_config_dir': '/etc/faucet', 'drop_lldp': True, 'learn_ban_timeout': 10, 'ofchannel_log': None, 'drop_broadcast_source_address': True, 'max_hosts_per_resolve_cycle': 5, 'proactive_learn': True, 'lldp_beacon': {}, 'cookie': 1524372928, 'stack': None, 'dp_id': 1, 'priority_offset': 0, 'description': 'sw1', 'max_host_fib_retry_count': 10, 'learn_jitter': 10, 'interfaces': {'p1': {'lldp_beacon': {}, 'unicast_flood': True, 'enabled': True, 'tagged_vlans': [], 'number': 1, 'description': 'port1', 'acls_in': None, 'mirror': None, 'acl_in': None, 'opstatus_reconf': True, 'hairpin': False, 'native_vlan': VLAN office vid:100 ports:Port 1,Port 2, 'loop_protect': False, 'stack': None, 'lacp': 0, 'override_output_port': None, 'receive_lldp': False, 'max_hosts': 255, 'permanent_learn': False, 'output_only': False}, 'p2': {'lldp_beacon': {}, 'unicast_flood': True, 'enabled': True, 'tagged_vlans': [], 'number': 2, 'description': 'port2', 'acls_in': None, 'mirror': None, 'acl_in': None, 'opstatus_reconf': True, 'hairpin': False, 'native_vlan': VLAN office vid:100 ports:Port 1,Port 2, 'loop_protect': False, 'stack': None, 'lacp': 0, 'override_output_port': None, 'receive_lldp': False, 'max_hosts': 255, 'permanent_learn': False, 'output_only': False}}, 'combinatorial_port_flood': True, 'packetin_pps': 0, 'ignore_learn_ins': 10, 'interface_ranges': {}, 'group_table_routing': False, 'advertise_interval': 30, 'group_table': False, 'low_priority': 9000, 'arp_neighbor_timeout': 250, 'drop_bpdu': True}
```

3. Reload faucet

To apply this configuration we can reload faucet which will cause it to compute the difference between the old and new configuration and apply the minimal set of changes to the network in a hitless fashion (where possible).

```
sudo systemctl reload faucet
```

4. Check logs

To verify the configuration reload was successful we can check `/var/log/faucet/faucet.log` and make sure faucet successfully loaded the configuration we can check the faucet log file `/var/log/faucet/faucet.log`:

Listing 4: `/var/log/faucet/faucet.log`

```
faucet INFO      Loaded configuration from /etc/faucet/faucet.yaml
faucet INFO      Add new datapath DPID 1 (0x1)
faucet INFO      Add new datapath DPID 2 (0x2)
faucet INFO      configuration /etc/faucet/faucet.yaml changed, analyzing_differences
faucet INFO      Reconfiguring existing datapath DPID 1 (0x1)
faucet.valve INFO DPID 1 (0x1) skipping configuration because datapath not up
faucet INFO      Deleting de-configured DPID 2 (0x2)
```

If there were any issues (say faucet wasn't able to find a valid pathway from the old config to the new config) we could issue a faucet restart now which will cause a cold restart of the network.

Configure gauge

We will not need to edit the default gauge configuration that is shipped with faucet as it will be good enough to complete the rest of this tutorial. If you did need to modify it the path is `/etc/faucet/gauge.yaml` and the default configuration looks like:

Listing 5: gauge.yaml

```
# Recommended configuration is Prometheus for all monitoring, with all_dps: True
faucet_configs:
  - '/etc/faucet/faucet.yaml'
watchers:
  port_status_poller:
    type: 'port_state'
    all_dps: True
    #dps: ['sw1', 'sw2']
    db: 'prometheus'
  port_stats_poller:
    type: 'port_stats'
    all_dps: True
    #dps: ['sw1', 'sw2']
    interval: 10
    db: 'prometheus'
    #db: 'influx'
  flow_table_poller:
    type: 'flow_table'
    all_dps: True
    interval: 60
    db: 'prometheus'
dbs:
  prometheus:
    type: 'prometheus'
    prometheus_addr: '0.0.0.0'
    prometheus_port: 9303
  ft_file:
    type: 'text'
    compress: True
    file: 'flow_table.yaml.gz'
  influx:
    type: 'influx'
    influx_db: 'faucet'
    influx_host: 'influxdb'
    influx_port: 8086
    influx_user: 'faucet'
    influx_pwd: 'faucet'
    influx_timeout: 10
```

This default configuration will setup a prometheus exporter listening on port 0.0.0.0:9303 and write all the different kind of gauge metrics to this exporter.

We will however need to restart the current gauge instance so it can pick up our new faucet configuration:

```
sudo systemctl restart gauge
```

Connect your first datapath

Now that we've set up all the different components let's connect our first switch (which we call a datapath) to faucet. We will be using [Open vSwitch](#) for this which is a production-grade software switch with very good OpenFlow support.

1. Add WAND Open vSwitch repo

The bundled version of Open vSwitch in Ubuntu 16.04 is quite old so we will use [WAND's package](#)

repo to install a newer version (if you're using a more recent debian or ubuntu release you can skip this step).

Note: If you're using a more recent debian or ubuntu release you can skip this step

```
sudo apt-get install apt-transport-https
echo "deb https://packages.wand.net.nz $(lsb_release -sc) main" | sudo_
tee /etc/apt/sources.list.d/wand.list
sudo curl https://packages.wand.net.nz/keyring.gpg -o /etc/apt/trusted.
gpg.d/wand.gpg
sudo apt-get update
```

2. Install Open vSwitch

```
sudo apt-get install openvswitch-switch
```

3. Add network namespaces to simulate hosts

We will use two linux network namespaces to simulate hosts and this will allow us to generate some traffic on our network.

First let's define some useful bash functions by coping and pasting the following definitions into our bash terminal:

```
create_ns () {
    NETNS=$1
    IP=$2
    sudo ip netns add ${NETNS}
    sudo ip link add dev veth-${NETNS} type veth peer name veth0 netns
    ↪$NETNS
    sudo ip link set dev veth-${NETNS} up
    sudo ip netns exec $NETNS ip link set dev veth0 up
    sudo ip netns exec $NETNS ip addr add dev veth0 $IP
    sudo ip netns exec $NETNS ip link set dev lo up
}

as_ns () {
    NETNS=$1
    shift
    sudo ip netns exec $NETNS $@
}
```

Now we will create host1 and host2 and assign them some IPs:

```
create_ns host1 192.168.0.1/24
create_ns host2 192.168.0.2/24
```

2. Configure Open vSwitch

We will now configure a single Open vSwitch bridge (which will act as our datapath) and add two ports to this bridge:

```
sudo ovs-vsctl add-br br0 \
-- set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 other-config:disable-in-band=true \
-- set bridge br0 fail_mode=secure \
-- add-port br0 veth-host1 -- set interface veth-host1 ofport_request=1 \
```

(continues on next page)

(continued from previous page)

```
-- add-port br0 veth-host2 -- set interface veth-host2 ofport_request=2 \  
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

The [Open vSwitch documentation](#) is very good if you wish to find out more about configuring Open vSwitch.

3. Verify datapath is connected to faucet

At this point everything should be working, we just need to verify that is the case. If we now load up some of the grafana dashboards we imported earlier, we should see the datapath is now listed in the `Faucet Inventory` dashboard.

If you don't see the new datapath listed you can look at the faucet log files `/var/log/faucet/faucet.log` or the Open vSwitch log `/var/log/openvswitch/ovs-vswitchd.log` for clues.

4. Generate traffic between virtual hosts

With `host1` and `host2` we can now test our network works and start generating some traffic which will show up in grafana.

Let's start simple with a ping:

```
as_ns host1 ping 192.168.0.2
```

If this test is successful this shows our Open vSwitch is forwarding traffic under faucet control, `/var/log/faucet/faucet.log` should now indicate those two hosts have been learnt:

Listing 6: `/var/log/faucet/faucet.log`

```
faucet.valve INFO      DPID 1 (0x1) L2 learned 22:a6:c7:20:ff:3b (L2 type_  
↪0x0806, L3 src 192.168.0.1, L3 dst 192.168.0.2) on Port 1 on VLAN 100_  
↪(1 hosts total)  
faucet.valve INFO      DPID 1 (0x1) L2 learned 36:dc:0e:b2:a3:4b (L2 type_  
↪0x0806, L3 src 192.168.0.2, L3 dst 192.168.0.1) on Port 2 on VLAN 100_  
↪(2 hosts total)
```

We can also use `iperf` to generate a large amount of traffic which will show up on the `Port Statistics` dashboard in grafana, just select `sw1` as the Datapath Name and `All` for the Port.

```
sudo apt-get install iperf3  
as_ns host1 iperf3 -s &  
as_ns host2 iperf3 -c 192.168.0.1
```

Further steps

Now that you know how to setup and run faucet in a self-contained virtual environment you can build on this tutorial and start to make more interesting topologies by adding more Open vSwitch bridges, ports and network namespaces. Check out the faucet [Configuration](#) document for more information on features you can turn on and off. In future we will publish additional tutorials on layer 3 routing, inter-vlan routing, ACLs.

You can also easily add real hardware into the mix as well instead of using a software switch. See the [Vendor-specific Documentation](#) section for information on how to configure a wide variety of different vendor devices for faucet.

1.2.2 ACL tutorial

In the *Installing faucet for the first time* tutorial we covered how to install and set-up Faucet. Next we are going to introduce Access Control Lists (ACLs).

ETA: ~25 minutes.

Prerequisites

- Install Faucet - *Package installation* steps 1 & 2
- Install Open vSwitch - *Connect your first datapath* steps 1 & 2
- Useful Bash Functions - Copy and paste the following definitions into your bash terminal, or to make them persistent between sessions add them to the bottom of your `.bashrc` and run `'source .bashrc'`.

```
create_ns () {
    NETNS=$1
    IP=$2
    sudo ip netns add ${NETNS}
    sudo ip link add dev veth-${NETNS} type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-${NETNS} up
    sudo ip netns exec $NETNS ip link set dev veth0 up
    sudo ip netns exec $NETNS ip addr add dev veth0 $IP
    sudo ip netns exec $NETNS ip link set dev lo up
}
```

```
as_ns () {
    NETNS=$1
    shift
    sudo ip netns exec $NETNS $@
}
```

```
cleanup () {
    for br in $(sudo ovs-vsctl list-br); do
        sudo ovs-vsctl del-br $br
    done
    IFS=$'\n';
    for i in $(sudo ip netns list); do
        NS=$(echo $i | cut -d ' ' -f1)
        sudo ip netns delete $NS;
        # Sometimes the veth-$NS is not deleted.
        sudo ip link delete veth-$NS;
    done
}
```

Note: If not continuing on from the 'Installing Faucet for first time tutorial' to setup the hosts and switch run:

```
create_ns host1 192.168.0.1/24
create_ns host2 192.168.0.2/24
sudo ovs-vsctl add-br br0 \
-- set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 other-config:disable-in-band=true \
-- set bridge br0 fail_mode=secure \
-- add-port br0 veth-host1 -- set interface veth-host1 ofport_request=1 \
```

(continues on next page)

(continued from previous page)

```
-- add-port br0 veth-host2 -- set interface veth-host2 ofport_request=2 \  
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

And the faucet.yaml configuration file looks like:

Listing 7: /etc/faucet/faucet.yaml

```
vlan:  
  office:  
    vid: 100  
    description: "office network"  
  
dps:  
  sw1:  
    dp_id: 0x1  
    hardware: "Open vSwitch"  
    interfaces:  
      1:  
        name: "host1"  
        description: "host2 network namespace"  
        native_vlan: office  
      2:  
        name: "host2"  
        description: "host2 network namespace"  
        native_vlan: office
```

Overview

Faucet ACLs are made up of lists of rules. The order of the rules in the list denote the priority with the first rules being highest and last lowest. Each of these lists has a name (e.g. 'block-ping'), and can be used on multiple port or VLAN 'acls_in' fields. Again these are applied in order so all of 'block-ping' rules will be higher than 'allow-all'.

Each rule contains two main items 'matches' and 'actions'. Matches are any packet field such as MAC/IP/transport source/destination fields. For a full list visit the [Ryu documentation](#).

Actions are used to control what the packet does, for example normal I2 forwarding ('allow'). Apply a 'meter' to rate limit traffic, and manipulation of the packet contents and output. Full list is available in the [Meters](#) section of the documentation.

The example below has defined two ACLs 'block-ping' & 'allow-all' these can be used on any and multiple ports or VLANs (more on VLANs later) using the 'acls_in' key. The block-ping ACL has two rules, one to block ICMP on IPv4 and another for ICMPv6 on IPv6. The allow-all ACL has one rule, which specifies no match fields, and therefore matches all packets, and the action 'allow'. The 'allow' action is a boolean, if it's True allow the packet to continue through the Faucet pipeline, if False drop the packet. 'allow' can be used in conjunction with the other actions to let the traffic flow with the expected layer 2 forwarding behaviour AND be mirrored to another port.

Network Setup

We are going to create the following network:

First we will add two new hosts to our network:


```
create_ns host3 192.168.0.3/24
create_ns host4 192.168.0.4/24
```

And connect them to br0

```
sudo ovs-vsctl add-port br0 veth-host3 -- set interface veth-host3 ofport_request=3 \
  -- add-port br0 veth-host4 -- set interface veth-host4 ofport_request=4
```

The configuration below will block ICMP on traffic coming in on port 3, and allow everything else. Add this to `/etc/faucet/faucet.yaml` below the ‘dps’.

Listing 8: `/etc/faucet/faucet.yaml`

```
3:
  name: "host3"
  native_vlan: office
  acls_in: [block-ping, allow-all]
4:
  name: "host4"
  native_vlan: office
acls:
  block-ping:
    - rule:
        dl_type: 0x800      # IPv4
        ip_proto: 1        # ICMP
        actions:
          allow: False
    - rule:
        dl_type: 0x86dd     # IPv6
        ip_proto: 58        # ICMPv6
        actions:
          allow: False
  allow-all:
    - rule:
        actions:
          allow: True
```

Now tell Faucet to reload its configuration, this can be done by restarting the application. But a better way is to send Faucet a SIGHUP signal.

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

```
sudo systemctl reload faucet
```

Now pings to/from host3 should fail, but the other three hosts should be fine.

Test this with

```
as_ns host1 ping 192.168.0.3
as_ns host1 ping 192.168.0.4
```

ACL Actions

Mirroring

Mirroring traffic is useful if we want to send it to an out of band NFV service (e.g. Intrusion Detection System, packet capture a port or VLAN). To do this Faucet provides two ACL actions: mirror & output.

The mirror action copies the packet, before any modifications, to the specified port.

Note: Mirroring is done in input direction only.

Let's add the mirror action to our block-ping ACL `/etc/faucet/faucet.yaml`

Listing 9: `/etc/faucet/faucet.yaml`

```
...
  block-ping:
    - rule:
        dl_type: 0x800
        ip_proto: 1
        actions:
            allow: False
            mirror: 4
    - rule:
        dl_type: 0x86dd
        ip_proto: 58
        actions:
            allow: False
            mirror: 4
```

And again send the sighup signal to Faucet

```
sudo systemctl reload faucet
```

To check this we will ping from host1 to host3, while performing a tcpdump on host4 who should receive the ping replies. It is a good idea to run each from a different terminal (screen, tmux, ...)

```
as_ns host4 tcpdump -l -e -n -i veth0
```

```
as_ns host1 ping 192.168.0.3
```

Ping should have 100% packet loss.

```
$ as_ns host4 tcpdump -l -e -n -i veth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:24:36.848331 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.3 > 192.168.0.1: ICMP echo reply, id 23660, seq 16, length 64
13:24:37.857024 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.3 > 192.168.0.1: ICMP echo reply, id 23660, seq 17, length 64
13:24:38.865005 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.3 > 192.168.0.1: ICMP echo reply, id 23660, seq 18, length 64
13:24:39.873377 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.3 > 192.168.0.1: ICMP echo reply, id 23660, seq 19, length 64
13:24:40.881129 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.3 > 192.168.0.1: ICMP echo reply, id 23660, seq 20, length 64
```

Output

There is also the ‘output’ action which can be used to achieve the same thing.

Listing 10: /etc/faucet/faucet.yaml

```
block-ping:
  - rule:
      dl_type: 0x800
      ip_proto: 1
      actions:
        allow: False
        output:
          port: 4
  - rule:
      dl_type: 0x86dd
      ip_proto: 58
      actions:
        allow: False
        output:
          port: 4
```

The output action also allows us to change the packet by setting fields (mac/ip addresses, ...), VLAN operations (push/pop/swap VIDs). It can be used in conjunction with the other actions, e.g. output directly but do not allow through the Faucet pipeline (allow: false).

Let’s create a new ACL for host2’s port that will change the MAC source address.

Listing 11: /etc/faucet/faucet.yaml

```
dps:
  sw1:
    ...
    2:
      name: "host2"
      description: "host2 network namespace"
      native_vlan: office
      acls_in: [rewrite-mac, allow-all]
    ...
acls:
  rewrite-mac:
    - rule:
        actions:
          allow: True
          output:
            set_fields:
              - eth_src: "00:00:00:00:00:02"
    ...
```

Again reload Faucet.

Start tcpdump on host1

```
as_ns host1 tcpdump -l -e -n -i veth0
```

Ping host1 from host2

```
as_ns host2 ping 192.168.0.1
```

Here we can see ICMP echo requests are coming from the MAC address “00:00:00:00:00:02” that we set in our output ACL. (The reply is destined to the actual MAC address of host2 thanks to ARP).

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:53:41.248235 00:00:00:00:00:02 > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.2 > 192.168.0.1: ICMP echo request, id 23711, seq 1, length 64
13:53:41.248283 06:5f:14:fc:47:02 > ce:bb:23:ce:d5:a0, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 23711, seq 1, length 64
13:53:42.247106 00:00:00:00:00:02 > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.2 > 192.168.0.1: ICMP echo request, id 23711, seq 2, length 64
13:53:42.247154 06:5f:14:fc:47:02 > ce:bb:23:ce:d5:a0, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 23711, seq 2, length 64
13:53:43.249726 00:00:00:00:00:02 > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.2 > 192.168.0.1: ICMP echo request, id 23711, seq 3, length 64
13:53:43.249757 06:5f:14:fc:47:02 > ce:bb:23:ce:d5:a0, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 23711, seq 3, length 64
13:53:44.248713 00:00:00:00:00:02 > 06:5f:14:fc:47:02, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.2 > 192.168.0.1: ICMP echo request, id 23711, seq 4, length 64
13:53:44.248738 06:5f:14:fc:47:02 > ce:bb:23:ce:d5:a0, ethertype IPv4 (0x0800),
↪length 98: 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 23711, seq 4, length 64
```

With the output action we could also use it to mirror traffic to a NFV server (like our fake mirror output action above), and use a VLAN tag to identify what port the traffic originated on on the switch. To do this we will use both the ‘port’ & ‘vlan_vid’ output fields.

Listing 12: /etc/faucet/faucet.yaml

```
block-ping:
  - rule:
      dl_type: 0x800
      ip_proto: 1
      actions:
        allow: False
        output:
          vlan_vid: 3
          port: 4
  - rule:
      dl_type: 0x86dd
      ip_proto: 58
      actions:
        allow: False
        output:
          vlan_vid: 3
          port: 4
```

Again reload Faucet, start a tcpdump on host4, and ping from host1 to host3. Ping should still not be allowed through and the tcpdump output should be similar to below (Note the 802.1Q tag and vlan 3):

```
$ as_ns host4 tcpdump -l -e -n -i veth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:14:15.285329 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype 802.1Q (0x8100),
↪length 102: vlan 3, p 0, ethertype IPv4, 192.168.0.3 > 192.168.0.1: ICMP echo reply,
↪id 23747, seq 1, length 64
14:14:16.293016 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype 802.1Q (0x8100),
↪length 102: vlan 3, p 0, ethertype IPv4, 192.168.0.3 > 192.168.0.1: ICMP echo reply,
↪id 23747, seq 2, length 64
```

(continues on next page)

(continued from previous page)

```
14:14:17.300898 2e:d4:1a:ca:54:4b > 06:5f:14:fc:47:02, ethertype 802.1Q (0x8100),
↳ length 102: vlan 3, p 0, ethertype IPv4, 192.168.0.3 > 192.168.0.1: ICMP echo reply,
↳ id 23747, seq 3, length 64
```

1.2.3 VLAN Tutorial

Next we are going to introduce VLANs.

ETA: ~30 mins.

Prerequisites

- Install Faucet - *Package installation* steps 1 & 2
- Install Open vSwitch - *Connect your first datapath* steps 1 & 2
- Useful Bash Functions - Copy and paste the following definitions into your bash terminal, or to make them persistent between sessions add them to the bottom of your `.bashrc` and run `'source .bashrc'`.

```
create_ns () {
    NETNS=$1
    IP=$2
    sudo ip netns add ${NETNS}
    sudo ip link add dev veth-${NETNS} type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-${NETNS} up
    sudo ip netns exec $NETNS ip link set dev veth0 up
    sudo ip netns exec $NETNS ip addr add dev veth0 $IP
    sudo ip netns exec $NETNS ip link set dev lo up
}
```

```
as_ns () {
    NETNS=$1
    shift
    sudo ip netns exec $NETNS $@
}
```

```
cleanup () {
    for br in $(sudo ovs-vsctl list-br); do
        sudo ovs-vsctl del-br $br
    done
    IFS=$'\n';
    for i in $(sudo ip netns list); do
        NS=$(echo $i | cut -d ' ' -f1)
        sudo ip netns delete $NS;
        # Sometimes the veth-$NS is not deleted.
        sudo ip link delete veth-$NS;
    done
}
```

- to add a tagged network interface to a host namespaces

```
add_tagged_dev_ns () {
    NETNS=$1
```

(continues on next page)

(continued from previous page)

```
IP=$2
VLAN=$3
sudo ip netns exec $NETNS ip link add link veth0 name veth0.${VLAN} type vlan id
↪$VLAN
sudo ip netns exec $NETNS ip link set dev veth0.${VLAN} up
sudo ip netns exec $NETNS ip addr flush dev veth0
sudo ip netns exec $NETNS ip addr add dev veth0.${VLAN} $IP
}
```

Overview

In this tutorial we will look at how to do the following tasks using Faucet:

- Use VLANs to segment traffic.
- Create Trunk links.
- ACLs for a particular VLAN.

Note: We cover *Routing between VLANs* in a later tutorial.

A port can be in several VLAN modes: 1. native - where packets come into the switch with no 802.1Q tag. 2. tagged - where packets come into the switch with a 802.1Q tag. 3. Both native and tagged.

If a packet comes in with a tag for a VLAN that the port is not configured for it will be dropped.

Configuring VLANs

To demonstrate these tasks we will use a demo network where a single switch br0 connects to 9 hosts.

Ports 1, 2, 5, 6 will be native (untagged) ports. While ports 3 & 4, 7 & 8, and 9 will be tagged ports.

Here is the structure of the demo setup.

Tip: Keep a piece of paper with the network layout and hosts' names, VLANs, IPs to simplify following the rest of the tutorial.

Network setup

Let's start. Keep host1, host2 on the native vlan 100 (office vlan) as in the first and second tutorials.

Note: To create the hosts and switch again run

```
cleanup
create_ns host1 192.168.0.1/24
create_ns host2 192.168.0.2/24
sudo ovs-vsctl add-br br0 \
-- set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 other-config:disable-in-band=true \
```

(continues on next page)

(continued from previous page)

```
-- set bridge br0 fail_mode=secure \
-- add-port br0 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- add-port br0 veth-host2 -- set interface veth-host2 ofport_request=2 \
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

Then add the following hosts with the corresponding vlan:

- Assign host3 and host4 a vlan interface (vid:100) as they are on a tagged port.
- Assign host5 and host6 an IP address from the VLAN 200 range.
- Assign host7 and host8 a vlan interface (vid:300) as they are on a tagged port.
- Add host9 to all vlans (100, 200, 300) to work as a NFV host.

Tagged vlan 100

```
create_ns host3 0
create_ns host4 0
add_tagged_dev_ns host3 192.168.0.3/24 100
add_tagged_dev_ns host4 192.168.0.4/24 100
```

Native vlan 200

```
create_ns host5 192.168.2.5/24
create_ns host6 192.168.2.6/24
```

Tagged vlan 300

```
create_ns host7 0
create_ns host8 0
add_tagged_dev_ns host7 192.168.3.7/24 300
add_tagged_dev_ns host8 192.168.3.8/24 300
```

Trunk link for host9

```
create_ns host9 0
add_tagged_dev_ns host9 192.168.0.9/24 100
add_tagged_dev_ns host9 192.168.2.9/24 200
add_tagged_dev_ns host9 192.168.3.9/24 300
```

Then connect all the hosts to the switch (br0)

```
sudo ovs-vsctl add-port br0 veth-host3 -- set interface veth-host3 ofport_request=3 \
-- add-port br0 veth-host4 -- set interface veth-host4 ofport_request=4 \
-- add-port br0 veth-host5 -- set interface veth-host5 ofport_request=5 \
-- add-port br0 veth-host6 -- set interface veth-host6 ofport_request=6 \
-- add-port br0 veth-host7 -- set interface veth-host7 ofport_request=7 \
-- add-port br0 veth-host8 -- set interface veth-host8 ofport_request=8 \
-- add-port br0 veth-host9 -- set interface veth-host9 ofport_request=9
```

Now we have everything to start working with faucet through its configuration file. Each time we will only need to change the configuration file and restart faucet (or send it HUP signal to reload the configuration file).

Basic vlan settings

Change /etc/faucet/faucet.yaml to reflect our setting.

Listing 13: /etc/faucet/faucet.yaml

```
vlan100:
  vid: 100
vlan200:
  vid: 200
vlan300:
  vid: 300
dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host2 network namespace"
        native_vlan: vlan100
      2:
        name: "host2"
        description: "host2 network namespace"
        native_vlan: vlan100
      3:
        name: "host3"
        tagged_vlans: [vlan100]
      4:
        name: "host4"
        tagged_vlans: [vlan100]
      5:
        name: "host5"
        native_vlan: vlan200
      6:
        name: "host6"
        native_vlan: vlan200
      7:
        name: "host7"
        tagged_vlans: [vlan300]
      8:
        name: "host8"
        tagged_vlans: [vlan300]
      9:
        name: "host9"
        tagged_vlans: [vlan100, vlan200, vlan300]
```

Send SIGHUP signal to reload the configuration file, and check how its log the new configuration in /var/log/faucet/faucet.log

```
sudo systemctl reload faucet
cat /var/log/faucet/faucet.log
```

Let's do the following simple tests:

1. Ping between hosts in the same vlan

```
as_ns host1 ping 192.168.0.2
as_ns host3 ping 192.168.0.4
as_ns host5 ping 192.168.2.6
as_ns host7 ping 192.168.3.8
```


All should work.

2. Ping between hosts in the same vlan where the port's vlan mode is both native and tagged. In particular between host1 (native vlan100) to host3 (tagged vlan100).

```
as_ns host1 ping 192.168.0.3
```

3. Ping between hosts in different vlans. Let's change host5 (native vlan200) ip to be 192.168.0.5 and try to ping it from host1 (native vlan100).

```
as_ns host5 ifconfig veth0 192.168.0.5
as_ns host1 ping 192.168.0.5
```

It will not ping as they are in different vlans. Let's set host5's IP back.

```
as_ns host5 ifconfig veth0 192.168.2.5
```

4. Test the trunk link to host9 from different vlans

```
as_ns host1 ping 192.168.0.9
as_ns host3 ping 192.168.0.9
as_ns host5 ping 192.168.2.9
as_ns host7 ping 192.168.3.9
```

All of this traffic should go through to host9 as it is connected through the trunk link.

Vlan ACL

Let's apply an ACL on a particular vlan (e.g. vlan300). We will block any ICMP packets on vlan300. First create an ACL to block the ping. Open /etc/faucet/faucet.yaml and add the 'acls' section.

Listing 14: /etc/faucet/faucet.yaml

```
acls:
  block-ping:
    - rule:
        dl_type: 0x800      # IPv4
        ip_proto: 1        # ICMP
        actions:
          allow: False
    - rule:
        dl_type: 0x86dd     # IPv6
        ip_proto: 58        # ICMPv6
        actions:
          allow: False
```

Then apply this ACL on vlan300.

Listing 15: /etc/faucet/faucet.yaml

```
vlans:
  vlan100:
    vid: 100
    faucet_vips: ["192.168.0.254/24"]
  vlan200:
    vid: 200
```

(continues on next page)

(continued from previous page)

```
faucet_vips: ["192.168.2.254/24"]
vlan300:
  vid: 300
  acls_in: [block-ping] # Acl apply only on vlan300
```

Just before we reload the configuration file. Let's verify that pinging is working between hosts in vlan300.

```
as_ns host7 ping 192.168.3.8
```

Now let's apply the configuration, send SIGHUP signal to reload the configuration file.

```
sudo systemctl reload faucet
```

Now if you try to ping from host7 and host8, it will not work as it is specified by their vlan acl.

```
as_ns host7 ping 192.168.3.8
```

1.2.4 Routing Tutorial

This tutorial will cover routing with Faucet.

There are three types of routing we can use.

- Inter VLAN routing
- Static routing
- BGP via an external application (Quagga, Bird, EXABGP, ...)

Prerequisites

- Install Faucet - *Package installation* steps 1 & 2
- Install Open vSwitch - *Connect your first datapath* steps 1 & 2
- Useful Bash Functions - Copy and paste the following definitions into your bash terminal, or to make them persistent between sessions add them to the bottom of your `.bashrc` and run 'source `.bashrc`'.

```
create_ns () {
  NETNS=$1
  IP=$2
  sudo ip netns add ${NETNS}
  sudo ip link add dev veth-${NETNS} type veth peer name veth0 netns $NETNS
  sudo ip link set dev veth-${NETNS} up
  sudo ip netns exec $NETNS ip link set dev veth0 up
  sudo ip netns exec $NETNS ip addr add dev veth0 $IP
  sudo ip netns exec $NETNS ip link set dev lo up
}
```

```
as_ns () {
  NETNS=$1
  shift
  sudo ip netns exec $NETNS $@
}
```

```
cleanup () {
    for br in $(sudo ovs-vsctl list-br); do
        sudo ovs-vsctl del-br $br
    done
    IFS=$'\n';
    for i in $(sudo ip netns list); do

        NS=$(echo $i | cut -d ' ' -f1)
        sudo ip netns delete $NS;
        # Sometimes the veth-$NS is not deleted.
        sudo ip link delete veth-$NS;
    done
}
```

Run the cleanup script to remove old namespaces and switches:

```
cleanup
```

Routing between VLANs

Let's start with a single switch connected to two hosts in two different vlans.

```
create_ns host1 10.0.0.1/24
create_ns host2 10.0.1.2/24
sudo ovs-vsctl add-br br1 \
-- set bridge br1 other-config:datapath-id=0000000000000001 \
-- set bridge br1 other-config:disable-in-band=true \
-- set bridge br1 fail_mode=secure \
-- add-port br1 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- add-port br1 veth-host2 -- set interface veth-host2 ofport_request=2 \
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

To allow traffic between vlans we use a router, and assign each VLAN at least one IP address (gateway IP address). Lets add the routers and vlans section like so.

Listing 16: /etc/faucet/faucet.yaml

```
vlans:
  vlan100:
    vid: 100
    faucet_vips: ["10.0.0.254/24"] # Faucet's virtual IP address for vlan100
  vlan200:
    vid: 200
    faucet_vips: ["10.0.1.254/24"] # Faucet's virtual IP address for vlan200

routers:
  router-1:
    vlans: [vlan100, vlan200] # names of vlans to allow routing between.

dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
```

(continues on next page)

(continued from previous page)

```
1:
  name: "host1"
  description: "host1 network namespace"
  native_vlan: vlan100
2:
  name: "host2"
  description: "host2 network namespace"
  native_vlan: vlan200
```

Send the SIGHUP signal to reload the configuration file.

```
sudo systemctl reload faucet
```

Add the default route to the 'faucet_vips' as above.

```
as_ns host1 ip route add default via 10.0.0.254 dev veth0
as_ns host2 ip route add default via 10.0.1.254 dev veth0
```

Then generate some traffic between our two hosts.

```
as_ns host1 ping 10.0.1.2
```

It should work and traffic should go through.

Static Routing

For this we will set-up a Faucet switch with three hosts. One of these hosts will act like a gateway,

Run the cleanup script to remove old namespaces and switches.

```
cleanup
```

Create 3 hosts, in 2 different subnets:

```
create_ns host1 10.0.0.1/24
create_ns host2 10.0.0.2/24
create_ns hostgw 10.0.1.3/24
```

And add a default route for each host to it's gateway router.

```
as_ns host1 ip route add default via 10.0.0.254
as_ns host2 ip route add default via 10.0.0.254
as_ns hostgw ip route add default via 10.0.1.254
```

Create the bridge and add hosts 1, 2 and the gw to br1.

```
sudo ovs-vsctl add-br br1 \
-- set bridge br1 other-config:datapath-id=0000000000000001 \
-- set bridge br1 other-config:disable-in-band=true \
-- set bridge br1 fail_mode=secure \
-- add-port br1 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- add-port br1 veth-host2 -- set interface veth-host2 ofport_request=2 \
-- add-port br1 veth-hostgw -- set interface veth-hostgw ofport_request=3 \
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

For this Faucet configuration we will start from scratch. First we need to define 2 VLANs.

1. Hosts.
2. Gateway.

Here we have 3 new options:

- `faucet_mac`: The MAC address of Faucet's routing interface on this VLAN. If we do not set `faucet_mac` for each VLAN, routed packets will be dropped unless `'drop_spoofed_faucet_mac'` is set to false.
- `faucet_vips`: The IP address for Faucet's routing interface on this VLAN. Multiple IP addresses (IPv4 & IPv6) can be used.
- `routes`: Static routes for this VLAN.

Listing 17: `/etc/faucet/faucet.yaml`

```
vlan:
  br1-hosts:
    vid: 100
    description: "h1 & h2's vlan"
    faucet_mac: "00:00:00:00:00:11"
    faucet_vips: ["10.0.0.254/24"]

  br1-gw:
    vid: 200
    description: "vlan for gw port"
    faucet_mac: "00:00:00:00:00:22"
    faucet_vips: ["10.0.1.254/24"]
    routes:
      - route:
          ip_dst: "0.0.0.0/24"
          ip_gw: '10.0.1.3'
```

As our routing interface is in a different VLAN, we will want to route between the two VLANs on the switch (br1-hosts & br1-peer). So as with inter VLAN routing we will create a router for each switch.

Listing 18: `/etc/faucet/faucet.yaml`

```
routers:
  router-br1:
    vlans: [br1-hosts, br1-gw]
```

And the rest of the configuration looks like this:

Listing 19: `/etc/faucet/faucet.yaml`

```
dps:
  br1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: br1-hosts
      2:
        name: "host2"
        description: "host2 network namespace"
```

(continues on next page)

(continued from previous page)

```
        native_vlan: br1-hosts
3:
    name: "gw:"
    description: "hostgw network namespace"
    native_vlan: br1-gw
```

Start/reload Faucet.

```
sudo systemctl restart faucet
```

And we should now be able to ping (the first few packets may get lost as ?arp? Does it's thing).

```
as_ns host1 ping 10.0.1.3
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=2 ttl=62 time=0.625 ms
64 bytes from 10.0.1.3: icmp_seq=3 ttl=62 time=0.133 ms
64 bytes from 10.0.1.3: icmp_seq=4 ttl=62 time=0.064 ms
64 bytes from 10.0.1.3: icmp_seq=5 ttl=62 time=0.090 ms
```

BGP Routing

For this section we are going to change our static routes from above into BGP routes.

BGP (and other routing) is provided by a NFV service, here we will use [BIRD](#). Other applications such as ExaBGP & Quagga could be used. Faucet imports all routes provided by this NFV service. This means we can use our service for other routing protocols (OSPF, RIP, etc) and apply filtering using the service's policy language.

Setup

To install BIRD:

```
sudo apt-get install bird
```

Our data plane will end up looking like below, you may notice how we have the Faucet application connected to the control plane and dataplane.

Note: When using BGP and Faucet, if changing Faucet's routing configuration (routers, static routes, or a VLAN's BGP configuration) the Faucet application must be restarted to reload the configuration correctly (not sighup reloaded).

Remove the static routes added above:

Listing 20: /etc/faucet/faucet.yaml

```
vlan:
  br1-hosts:
    vid: 100
    description: "h1 & h2's vlan"
    faucet_mac: "00:00:00:00:00:11"
    faucet_vips: ["10.0.0.254/24"]
```

(continues on next page)

(continued from previous page)

```

br1-gw:
  vid: 200
  description: "vlan for peering port"
  faucet_mac: "00:00:00:00:00:22"
  faucet_vips: ["10.0.1.254/24"]

routers:
  router-br1:
    vlans: [br1-hosts, br1-gw]

dps:
  br1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: br1-hosts
      2:
        name: "host2"
        description: "host2 network namespace"
        native_vlan: br1-hosts
      3:
        name: "gw"
        description: "hostgw network namespace"
        native_vlan: br1-gw

```

Reload Faucet

```
sudo systemctl reload faucet
```

And check that host1 can ping host2 but not the gw.

```
as_ns host1 ping 10.0.0.2
as_ns host1 ping 10.0.1.3
```

Next we will add Faucet to our switch's data plane so that it can communicate with the BGP speaking hostgw.

```

sudo ip link add veth-faucet0 type veth peer name veth-faucet-dp
sudo ovs-vsctl add-port br1 veth-faucet-dp -- set interface veth-faucet-dp ofport_
↪request=4
sudo ip addr add 10.0.1.4/24 dev veth-faucet0
sudo ip link set veth-faucet0 up
sudo ip link set veth-faucet-dp up

```

We will also add another host connected to hostgw to act as the Internet and give it the IP address 1.0.0.1.

```

create_ns hostwww 172.16.0.1/24
as_ns hostwww ip route add default via 172.16.0.2
as_ns hostwww ip addr add 1.0.0.1/24 dev veth0
sudo ip link set veth-hostwww netns hostgw
as_ns hostgw ip addr add 172.16.0.2/24 dev veth-hostwww
as_ns hostgw ip link set veth-hostwww up
as_ns hostgw ip route replace default via 172.16.0.1
as_ns hostgw ip route add 10.0.0.0/24 via 10.0.1.254

```

To configure BIRD

Listing 21: /etc/bird/bird.conf

```
protocol kernel {
    scan time 60;
    import none;
}

protocol device {
    scan time 60;
}

# Local
protocol static {
    route 172.16.0.0/24 via 172.16.0.2
    route 1.0.0.0/24 via 172.16.0.2
}

# Faucet bgp peer config.
# Will import all routes available, including the static ones above.
protocol bgp faucet {
    local as 64513;
    neighbor 10.0.1.4 port 9179 as 64512;
    export all;
    import all;
    next hop self;
}
```

Create the directory for Bird's server control socket and start BIRD:

```
sudo mkdir /run/bird
as_ns hostgw bird
```

We'll configure Faucet by adding the BGP configuration to the br1-gw VLAN.

Listing 22: /etc/faucet/faucet.yaml

```
vlan:
  br1-hosts:
    vid: 100
    description: "h1 & h2's vlan"
    faucet_mac: "00:00:00:00:00:11"
    faucet_vips: ["10.0.0.254/24"]

  br1-gw:
    vid: 200
    description: "vlan for peering port"
    faucet_mac: "00:00:00:00:00:22"
    faucet_vips: ["10.0.1.254/24"]
    bgp_port: 9179
    bgp_as: 64512
    bgp_routerid: '10.0.1.4'
    bgp_neighbor_addresses: ['10.0.1.3']
    ↪ IPv6)
    bgp_connect_mode: active
    bgp_neighbor_as: 64513

# BGP port for Faucet to listen on.
# Faucet's AS number
# Faucet's Unique ID.
# Neighbouring IP addresses (IPv4/
#
# Neighbour's AS number
```

(continues on next page)

(continued from previous page)

```

routers:
  br1-router:
    vlans: [br1-hosts, br1-gw]

```

And finally add the port configuration for the Faucet data plane interface (veth-faucet0).

Listing 23: /etc/faucet/faucet.yaml

```

dps:
  br1:
    ...
    interfaces:
      ...
      4:
        name: "faucet-dataplane"
        description: "faucet's dataplane connection for bgp"
        native_vlan: br1-gw

```

Now restart Faucet.

```
sudo systemctl restart faucet
```

and our logs should show us ‘BGP peer router ID 10.0.1.3 AS 64513 up’ & ‘BGP add 172.16.0.0/24 nexthop 10.0.1.3’ which is our route advertised via BGP.

Listing 24: /var/log/faucet/faucet.log

```

May 10 13:42:54 faucet INFO      Reloading configuration
May 10 13:42:54 faucet INFO      configuration /etc/faucet/faucet.yaml changed,
↪analyzing differences
May 10 13:42:54 faucet INFO      Add new datapath DPID 1 (0x1)
May 10 13:42:55 faucet INFO      BGP peer router ID 10.0.1.3 AS 64513 up
May 10 13:42:55 faucet INFO      BGP add 172.16.0.0/24 nexthop 10.0.1.3
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Cold start configuring DP
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Configuring VLAN br1-gw vid:200
↪ports:Port 3,Port 4
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Configuring VLAN br1-hosts vid:100
↪ports:Port 1,Port 2
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Port 1 configured
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Port 2 configured
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Port 3 configured
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Port 4 configured
May 10 13:42:55 faucet.valve INFO DPID 1 (0x1) Ignoring port:4294967294 not
↪present in configuration file
May 10 13:42:56 faucet.valve INFO DPID 1 (0x1) resolving 10.0.1.3 (2 flows) on
↪VLAN 200

```

Now we should be able to ping from host1 to hostwww.

```

as_ns host1 ping 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=2 ttl=62 time=0.165 ms
64 bytes from 172.16.0.1: icmp_seq=3 ttl=62 time=0.058 ms
64 bytes from 172.16.0.1: icmp_seq=4 ttl=62 time=0.057 ms

```

```
as_ns host1 ping 1.0.0.1
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=62 time=0.199 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=62 time=0.053 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=62 time=0.058 ms
64 bytes from 1.0.0.1: icmp_seq=4 ttl=62 time=0.054 ms
```

1.2.5 NFV Services Tutorial

This tutorial will cover using faucet with Network Function Virtualisation (NFV) services.

NFV services that will be demonstrated in this tutorial are:

- DHCP server
- NAT Gateway
- **BRO** Intrusion Detection System (IDS)

This tutorial demonstrates how the previous topics in this tutorial series can be integrated with other services on our network.

Prerequisites

- A good understanding of the previous tutorial topics (*ACL tutorial*, *VLAN Tutorial*, *Routing Tutorial*)
- Install Faucet - *Package installation* steps 1 & 2
- Install Open vSwitch - *Connect your first datapath* steps 1 & 2
- Useful Bash Functions - Copy and paste the following definitions into your bash terminal, or to make them persistent between sessions add them to the bottom of your `.bashrc` and run `'source .bashrc'`.

```
create_ns () {
    NETNS=$1
    IP=$2
    sudo ip netns add ${NETNS}
    sudo ip link add dev veth-${NETNS} type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-${NETNS} up
    sudo ip netns exec $NETNS ip link set dev veth0 up
    sudo ip netns exec $NETNS ip addr add dev veth0 $IP
    sudo ip netns exec $NETNS ip link set dev lo up
}
```

```
as_ns () {
    NETNS=$1
    shift
    sudo ip netns exec $NETNS $@
}
```

```
cleanup () {
    for br in $(sudo ovs-vsctl list-br); do
        sudo ovs-vsctl del-br $br
    done
    IFS=$'\n';
    for i in $(sudo ip netns list); do
```

(continues on next page)

(continued from previous page)

```

    NS=$(echo $i | cut -d ' ' -f1)
    sudo ip netns delete $NS;
    # Sometimes the veth-$NS is not deleted.
    sudo ip link delete veth-$NS;
done
}

```

```

add_tagged_dev_ns () {
    NETNS=$1
    IP=$2
    VLAN=$3
    sudo ip netns exec $NETNS ip link add link veth0 name veth0.${VLAN} type vlan id
↪$VLAN
    sudo ip netns exec $NETNS ip link set dev veth0.${VLAN} up
    sudo ip netns exec $NETNS ip addr flush dev veth0
    sudo ip netns exec $NETNS ip addr add dev veth0.${VLAN} $IP
}

```

```

clear_ns() {
    NETNS=$1
    sudo ip netns delete $NETNS
    sudo ovs-vsctl del-port br0 veth-${NETNS}
}

```

Let's start by run the cleanup script to remove old namespaces and switches.

```
cleanup
```

Network setup

We will create a switch and attach seven hosts like so:

The network will be divided into 3 VLANs like so, 2 client vlans (200 & 300) with two clients each (host4-7), and a VLAN for the Bro server. The layer 2 & 3 diagram looks like:

```

# BRO
create_ns host1 192.168.0.1/24

# DHCP server
create_ns host2 0
add_tagged_dev_ns host2 192.168.2.2/24 200 # to serve vlan 200
add_tagged_dev_ns host2 192.168.3.2/24 300 # to serve vlan 300

# Gateway
create_ns host3 0
add_tagged_dev_ns host3 192.168.2.3/24 200 # to serve vlan 200
add_tagged_dev_ns host3 192.168.3.3/24 300 # to serve vlan 200

# vlan 200 hosts
create_ns host4 0

```

(continues on next page)

(continued from previous page)

```
create_ns host5 0
# vlan 300 hosts
create_ns host6 0
create_ns host7 0
```

Then create an OpenvSwitch and connect all hosts to it.

```
sudo ovs-vsctl add-br br0 \
-- set bridge br0 other-config:datapath-id=0000000000000001 \
-- set bridge br0 other-config:disable-in-band=true \
-- set bridge br0 fail_mode=secure \
-- add-port br0 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- add-port br0 veth-host2 -- set interface veth-host2 ofport_request=2 \
-- add-port br0 veth-host3 -- set interface veth-host3 ofport_request=3 \
-- add-port br0 veth-host4 -- set interface veth-host4 ofport_request=4 \
-- add-port br0 veth-host5 -- set interface veth-host5 ofport_request=5 \
-- add-port br0 veth-host6 -- set interface veth-host6 ofport_request=6 \
-- add-port br0 veth-host7 -- set interface veth-host7 ofport_request=7 \
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

DHCP Server

We will use [dnsmasq](#) as our DHCP server.

First install dnsmasq:

```
sudo apt-get install dnsmasq
```

Let's run two services one for vlan 200 and another for vlan 300 as following

```
# 192.168.2.0/24 for vlan 200
as_ns host2 dnsmasq --no-ping -p 0 -k \
--dhcp-range=192.168.2.10,192.168.2.20 \
--dhcp-sequential-ip \
--dhcp-option=option:router,192.168.2.3 \
-O option:dns-server,8.8.8.8 \
-I lo -z -l /tmp/nfv-dhcp-vlan200.leases \
-8 /tmp/nfv.dhcp-vlan200.log -i veth0.200 --conf-file= &

# 192.168.3.0/24 for vlan 300
as_ns host2 dnsmasq --no-ping -p 0 -k \
--dhcp-range=192.168.3.10,192.168.3.20 \
--dhcp-sequential-ip \
--dhcp-option=option:router,192.168.3.3 \
-O option:dns-server,8.8.8.8 \
-I lo -z -l /tmp/nfv-dhcp-vlan300.leases \
-8 /tmp/nfv.dhcp-vlan300.log -i veth0.300 --conf-file= &
```

Now let's configure faucet yaml file (/etc/faucet/faucet.yaml)

Listing 25: /etc/faucet/faucet.yaml

```
vlan:
  bro-vlan:
    vid: 100
```

(continues on next page)

(continued from previous page)

```

        description: "bro network"
    vlan200:
        vid: 200
        description: "192.168.2.0/24 network"
    vlan300:
        vid: 300
        description: "192.168.3.0/24 network"
dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "BRO network namespace"
                native_vlan: bro-vlan
            2:
                name: "host2"
                description: "DHCP server network namespace"
                tagged_vlans: [vlan200, vlan300]
            3:
                name: "host3"
                description: "gateway network namespace"
                tagged_vlans: [vlan200, vlan300]
            4:
                name: "host4"
                description: "host4 network namespace"
                native_vlan: vlan200
            5:
                name: "host5"
                description: "host5 network namespace"
                native_vlan: vlan200
            6:
                name: "host6"
                description: "host6 network namespace"
                native_vlan: vlan300
            7:
                name: "host7"
                description: "host7 network namespace"
                native_vlan: vlan300

```

Now reload faucet configuration file.

```
sudo systemctl reload faucet
```

Use dhclient to configure host4 to host7 using DHCP (it may take a few seconds, but should return when successful).

```

as_ns host4 dhclient veth0
as_ns host5 dhclient veth0
as_ns host6 dhclient veth0
as_ns host7 dhclient veth0

```

You can check `/tmp/nfv-dhcp<vlanid>.leases` and `/tmp/nfv.dhcp<vlanid>.log` to find their IPs. e.g. file `/tmp/nfv-dhcp-vlan300.leases`

Listing 26: output:

```
1525938604 7e:bb:f0:46:6a:e8 192.168.3.11 ubuntu *
1525938567 76:58:6c:26:78:44 192.168.3.10 * *
```

Alternatively:

```
as_ns host4 ip addr show
as_ns host5 ip addr show
as_ns host6 ip addr show
as_ns host7 ip addr show
```

If the hosts have IPs then great our DHCP works,

Try to ping between them

```
as_ns host4 ping <ip of host5> # both in vlan200 should work
as_ns host6 ping <ip of host7> # both in vlan300 should work
as_ns host4 ping <ip of host6> # each in different vlan should not work
```

Ping between hosts vlan 200 and vlan 300 works because host3 (gateway) forwards the traffic by default. So we will fix this for the next sections by changing iptables on host3 to not route traffic by default.

```
as_ns host3 iptables -P FORWARD DROP
```

Now the ping should fail

```
as_ns host4 ping <host7 ip addr>
```

Gateway (NAT)

In this section we will configure host3 as a gateway (NAT) to provide internet connection for our network.

Listing 27: Define some variables we will use:

```
NS=host3          # gateway host namespace
TO_DEF=to_def     # to the internet
TO_NS=to_${NS}    # to gw (host3)
OUT_INTF=enp0s3   # host machine interface for internet connection.
```

Listing 28: Enable forwarding in the hosted machine and in the host3 namespace.

```
sudo sysctl net.ipv4.ip_forward=1
as_ns ${NS} sysctl net.ipv4.ip_forward=1
```

Listing 29: Create the link to bridge the two namespaces

```
sudo ip link add name ${TO_NS} type veth peer name ${TO_DEF} netns ${NS}
sudo ip addr add 192.168.100.1/30 dev ${TO_NS}
sudo ip link set ${TO_NS} up
```

Listing 30: Configure link towards the root namespace on the GW host's namespace.

```
as_ns ${NS} ip addr add 192.168.100.2/30 dev ${TO_DEF}
as_ns ${NS} ip link set ${TO_DEF} up
as_ns ${NS} ip route add default via 192.168.100.1
```

Listing 31: Do not allow routing between vlan300 & vlan200 on the gateway host. Allow each vlan to be sent to/from the gateway interface while being NAT-ed.

```
as_ns ${NS} iptables -P FORWARD DROP

as_ns ${NS} iptables -A FORWARD -i veth0.200 -o ${TO_DEF} -j ACCEPT
as_ns ${NS} iptables -A FORWARD -i veth0.300 -o ${TO_DEF} -j ACCEPT
as_ns ${NS} iptables -A FORWARD -i ${TO_DEF} -o veth0.200 -j ACCEPT
as_ns ${NS} iptables -A FORWARD -i ${TO_DEF} -o veth0.300 -j ACCEPT

as_ns ${NS} iptables -t nat -F
as_ns ${NS} iptables -t nat -A POSTROUTING -o ${TO_DEF} -j MASQUERADE
```

Listing 32: Assuming the host does not have other NAT rules. Setup the final forward towards the internet.

```
sudo iptables -P FORWARD DROP
sudo iptables -F FORWARD
sudo iptables -t nat -F
sudo iptables -t nat -A POSTROUTING -s 192.168.100.0/30 -o ${OUT_INTF} -j MASQUERADE
sudo iptables -A FORWARD -i ${OUT_INTF} -o ${TO_NS} -j ACCEPT
sudo iptables -A FORWARD -i ${TO_NS} -o ${OUT_INTF} -j ACCEPT
```

Now try to ping google.com from any host, it should work as the gateway and DNS is now configured.

```
as_ns host4 ping www.google.com
as_ns host7 ping www.google.com
```

BRO IDS

BRO installation

We need first to install bro. We will use the binary package version 2.5.3 for this test.

```
sudo apt-get install bro broctl
```

Configure BRO

In /etc/bro/node.cfg, set veth0 as the interface to monitor

Listing 33: /etc/bro/node.cfg

```
[bro]
type=standalone
```

(continues on next page)

(continued from previous page)

```
host=localhost
interface=veth0
```

Comment out MailTo in /etc/bro/broctl.cfg

Listing 34: /etc/bro/broctl.cfg

```
# Recipient address for all emails sent out by Bro and BroControl.
# MailTo = root@localhost
```

Run bro in host2

Since this is the first-time use of the bro command shell application, perform an initial installation of the BroControl configuration:

```
as_ns host1 broctl install
```

Then start bro instant

```
as_ns host1 broctl start
```

Check bro status

```
as_ns host1 broctl status
```

Name	Type	Host	Status	Pid	Started
bro	standalone	localhost	running	15052	07 May 09:03:59

Now let's add a mirror ACL so all vlan200 & vlan300 traffic is sent to BRO.

We will use vlan acls (more about acl and vlan check vlan and acl tutorials).

Listing 35: /etc/faucet/faucet.yaml

```
acls:
  mirror-acl:
    - rule:
        actions:
          allow: true
          mirror: 1
vlangs:
  bro-vlan:
    vid: 100
    description: "bro network"
  vlan200:
    vid: 200
    description: "192.168.2.0/24 network"
    acls_in: [mirror-acl]
  vlan300:
    vid: 300
    description: "192.168.3.0/24 network"
    acls_in: [mirror-acl]
dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
```

(continues on next page)

(continued from previous page)

```

interfaces:
  1:
    name: "host1"
    description: "BRO network namespace"
    native_vlan: bro-vlan
  2:
    name: "host2"
    description: "DHCP server network namespace"
    tagged_vlans: [vlan200, vlan300]
  3:
    name: "host3"
    description: "gateway network namespace"
    tagged_vlans: [vlan200, vlan300]
  4:
    name: "host4"
    description: "host4 network namespace"
    native_vlan: vlan200
  5:
    name: "host5"
    description: "host5 network namespace"
    native_vlan: vlan200
  6:
    name: "host6"
    description: "host6 network namespace"
    native_vlan: vlan300
  7:
    name: "host7"
    description: "host7 network namespace"
    native_vlan: vlan300

```

As usual reload faucet configuration file.

```
sudo systemctl reload faucet
```

If we generate some DHCP traffic on either of the hosts VLANs

```
as_ns host4 dhclient veth0
```

Then if we inspect the bro logs, we should see that bro has learnt about the two DHCP Servers. If the file does not exist check that faucet has successfully reloaded, and try the dhclient command again.

```
sudo cat /var/log/bro/current/known_services.log
```

Listing 36: output:

```

#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path known_services
#open 2018-05-10-12-09-05
#fields ts      host      port_num      port_proto      service
#types time     addr      port  enum      set[string]
1525910945.405356 192.168.3.2  67      udp      DHCP
1525910975.329404 192.168.2.2  67      udp      DHCP

```

1.3 Installation

1.3.1 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 directories for FAUCET and Gauge to log to.

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

```
vlan:
  100:
    description: "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 make the results available to Prometheus. See [Configuration](#) for more advanced configuration.

```
faucet_configs:
  - '/etc/faucet/faucet.yaml'
watchers:
  port_stats:
    dps: ['switch-1']
    type: 'port_stats'
    interval: 10
    db: 'prometheus'
  flow_table:
    dps: ['switch-1']
    type: 'flow_table'
    interval: 10
    db: 'prometheus'
dbs:
  prometheus:
    type: 'prometheus'
    prometheus_port: 9303
    prometheus_addr: ''
```

1.3.2 Installation using APT

We maintain a apt repo for installing faucet and its dependencies on Debian based Linux distributions.

Here is a list of packages we supply:

Package	Description
python3-faucet	Install standalone faucet/gauge python3 library
faucet	Install python3 library, systemd service and default config files
gauge	Install python3 library, systemd service and default config files
faucet-all-in-one	Install faucet, gauge, prometheus and grafana. Easy to use and good for testing faucet for the first time.

Installation on Debian 8 (jessie)

Installing faucet on jessie requires jessie-backports.

First follow the [official instructions](#) on adding the backports repo to jessie.

```
sudo apt-get install curl apt-transport-https gnupg lsb-release
echo "deb https://packagecloud.io/faucetsdn/faucet/$(lsb_release -si | awk '{print_t_
↳ tolower($0)}')/ $(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/faucet.
↳ list
curl -L https://packagecloud.io/faucetsdn/faucet/gpgkey | sudo apt-key add -
sudo apt-get install -t jessie-backports python3-oslo.config libjs-jquery libjs-
↳ mustache
sudo apt-get update
```

Then to install all components for a fully functioning system on a single machine:

```
sudo apt-get install faucet-all-in-one
```

or you can install the individual components:

```
sudo apt-get install faucet
sudo apt-get install gauge
```

Installation on Debian 9+ and Ubuntu 16.04+

```
sudo apt-get install curl gnupg apt-transport-https lsb-release
echo "deb https://packagecloud.io/faucetsdn/faucet/$(lsb_release -si | awk '{print_t_
↳ tolower($0)}')/ $(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/faucet.
↳ list
curl -L https://packagecloud.io/faucetsdn/faucet/gpgkey | sudo apt-key add -
sudo apt-get update
```

Then to install all components for a fully functioning system on a single machine:

```
sudo apt-get install faucet-all-in-one
```

or you can install the individual components:

```
sudo apt-get install faucet
sudo apt-get install gauge
```

1.3.3 Installation with Docker

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 our [Docker](#) section for details on how to install and start the Faucet and Gauge docker images.

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

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

1.3.4 Installation with pip

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

First, install some python dependencies:

```
apt-get install python3-dev python3-pip
pip3 install setuptools
pip3 install wheel
```

Then install the latest stable release of faucet from pypi, via pip:

```
pip3 install faucet
```

Or, install the latest development code from git, via pip:

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

Starting Faucet Manually

Faucet includes a start up script for starting Faucet and Gauge easily from the command line.

To run Faucet manually:

```
faucet --verbose
```

To run Gauge manually:

```
gauge --verbose
```

There are a number of options that you can supply the start up script for changing various options such as OpenFlow port and setting up an encrypted control channel. You can find a list of the additional arguments by running:

```
faucet --help
```

Starting Faucet With Systemd

Systemd can be used to start Faucet and Gauge at boot 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 37: 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/faucet --ryu-config-file=${FAUCET_RYU_CONF} --ryu-ofp-tcp-
↳listen-port=${FAUCET_LISTEN_PORT}
ExecReload=/bin/kill -HUP $MAINPID
Restart=always

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

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

Listing 38: 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/gauge --ryu-config-file=${GAUGE_RYU_CONF} --ryu-ofp-tcp-
↳listen-port=${GAUGE_LISTEN_PORT} --ryu-wsapi-host=${WSAPI_LISTEN_HOST} --ryu-
↳app=ryu.app.ofctl_rest
Restart=always

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

1.3.5 Virtual Machine Image

We provide a VM image for running FAUCET for development and learning purposes. The VM comes pre-installed with FAUCET, GAUGE, prometheus and grafana.

Openstack's [diskimage-builder](#) (DIB) is used to build the VM images in many formats (qcow2,tgz,squashfs,vhd,raw).

Pre-built images are available on our build host <https://builder.faucet.nz>.

Building the images

If you don't want to use our [pre-built images](#), you can build them yourself:

1. [Install the latest disk-image-builder](#)

2. [Install a patched vhd-util](#)
3. Run `build-faucet-vm.sh`

Security Considerations

This VM is not secure by default, it includes no firewall and has a number of network services listening on all interfaces with weak passwords. It also includes a backdoor user (faucet) with weak credentials.

Services

The VM exposes a number of ports listening on all interfaces by default:

Service	Port
SSH	22
Faucet OpenFlow Channel	6653
Gauge OpenFlow Channel	6654
Grafana Web Interface	3000
Prometheus Web Interface	9090

Default Credentials

Service	Username	Password
VM TTY Console	faucet	faucet
SSH	faucet	faucet
Grafana Web Interface	admin	admin

Post-Install Steps

Grafana comes installed but unconfigured, you will need to login to the grafana web interface at `http://VM_IP:3000` and configure a data source and some dashboards.

After logging in with the default credentials shown above, the first step is to add a [prometheus data source](#), use the following settings then click `Save & Test`:

Name:	Prometheus
Type:	Prometheus
URL:	<code>http://localhost:9090</code>

Next we want to add some dashboards so that we can later view the metrics from faucet.

Hover over the `+` button on the left sidebar in the web interface and click `Import`.

We will import the following dashboards, just download the following links and upload them through the grafana dashboard import screen:

- [Instrumentation](#)
- [Inventory](#)
- [Port Statistics](#)

You will need to supply your own `faucet.yaml` and `gauge.yaml` configuration in the VM. There are samples provided at `/etc/faucet/faucet.yaml` and `/etc/faucet/gauge.yaml`.

Finally you will need to point one of the supported OpenFlow vendors at the controller VM, port 6653 is the Faucet OpenFlow control channel and 6654 is the Gauge OpenFlow control channel for monitoring.

1.4 Docker

1.4.1 Installing docker

We recommend installing Docker Community Edition (CE) according to the official [docker engine installation guide](#).

1.4.2 Initial configuration

```
sudo mkdir -p /etc/faucet
sudo vi /etc/faucet/faucet.yaml
sudo vi /etc/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/faucet` to configure the switch pipeline.

1.4.3 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 stable release of Faucet. All tagged versions of Faucet in git are also available to use, for example using the `faucet/faucet:1.8.0` Docker will run the released version 1.8.0 of Faucet.

By default the Faucet and Gauge images are run as the `faucet` user under UID 0, GID 0. If you need to change that it can be overridden at runtime with the Docker flags: `-e LOCAL_USER_ID` and `-e LOCAL_GROUP_ID`.

To pull and run the latest version of Faucet:

```
mkdir -p /var/log/faucet/
docker pull faucet/faucet:latest
docker run -d \
  --name faucet \
  --restart=always \
  -v /etc/faucet:/etc/faucet/ \
  -v /var/log/faucet:/var/log/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 version of Gauge:

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

(continues on next page)

(continued from previous page)

```
-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.

1.4.4 Additional Arguments

You may wish to run faucet under docker with additional arguments, for example: setting certificates for an encrypted control channel. This can be done by overriding the docker entrypoint like so:

```
docker run -d \
  --name faucet \
  --restart=always \
  -v /etc/faucet:/etc/faucet/ \
  -v /etc/ryu/ssl:/etc/ryu/ssl/ \
  -v /var/log/faucet:/var/log/faucet/ \
  -p 6653:6653 \
  -p 9302:9302 \
  faucet/faucet \
  faucet \
  --ctl-privkey /etc/ryu/ssl/ctrlr.key \
  --ctl-cert /etc/ryu/ssl/ctrlr.cert \
  --ca-certs /etc/ryu/ssl/sw.cert
```

You can get a list of all additional arguments faucet supports by running:

```
docker run -it faucet/faucet faucet --help
```

1.4.5 Docker compose

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 pull
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.yml` 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 variable 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
```

And the following settings for InfluxDB:

```
Name: InfluxDB
Type: InfluxDB
Url: http://influxdb:8086
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. Hover over the + button on the left sidebar in the web interface and click Import.

We will import the following dashboards, just download the following links and upload them through the grafana dashboard import screen:

- [Instrumentation](#)
- [Inventory](#)
- [Port Statistics](#)

1.5 Configuration

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

Listing 39: faucet.yaml

```
include:
  - acfs.yaml

vlangs:
  office:
    vid: 100
    description: "office network"
    acfs_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']
```

(continues on next page)

(continued from previous page)

```
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
        acls_in: [access-port-protect]
      2:
        name: "h2"
        description: "host2 container"
        native_vlan: office
        acls_in: [access-port-protect]
      3:
        name: "g1"
        description: "guest1 container"
        native_vlan: guest
        acls_in: [access-port-protect]
      4:
        name: "s1"
        description: "services container"
        native_vlan: office
        acls_in: [service-port-protect]
      5:
        name: "trunk"
        description: "VLAN trunk to sw2"
        tagged_vlans: [office]
        acls_in: [access-port-protect]
  sw2:
    dp_id: 0x2
    hardware: "Allied-Telesis"
    interfaces:
      1:
        name: "pi"
        description: "raspberrypi"
        native_vlan: office
        acls_in: [access-port-protect]
      2:
        name: "laptop"
        description: "guest laptop"
        native_vlan: guest
        acls_in: [access-port-protect]
      3:
        name: "s1"
        description: "services Laptop"
        native_vlan: guest
        acls_in: [access-port-protect]
      24:
        name: "trunk"
        description: "VLAN trunk to sw1"
```

(continues on next page)

(continued from previous page)

```
tagged_vlans: [office, guest]
```

Listing 40: acls.yaml

```
acls:
  office-vlan-protect:
    # Prevent IPv4 communication between Office/Guest networks
    - rule:
        dl_type: 0x800      # ipv4
        ipv4_src: 10.0.100.0/24
        ipv4_dst: 10.0.200.0/24
        actions:
            allow: 0        # drop
    - rule:
        actions:
            allow: 1        # allow

  access-port-protect:
    # Drop dhcp servers
    - rule:
        dl_type: 0x800      # ipv4
        nw_proto: 17        # udp
        udp_src: 67         # bootps
        udp_dst: 68         # bootpc
        actions:
            allow: 0        # drop
    # Drop dhcpv6 servers
    - rule:
        dl_type: 0x86dd     # ipv6
        nw_proto: 17        # udp
        udp_src: 547        # dhcpv6-server
        udp_dst: 546        # dhcpv6-client
        actions:
            allow: 0        # drop
    # Drop icmpv6 RAs
    - rule:
        dl_type: 0x86dd     # ipv6
        nw_proto: 58        # icmpv6
        icmpv6_type: 134    # router advertisement
        actions:
            allow: 0        # drop
    # Drop SMTP
    - rule:
        dl_type: 0x800      # ipv4
        nw_proto: 6         # tcp
        tcp_dst: 25         # smtp
        actions:
            allow: 0        # drop
    # Force DNS to our DNS server
    - rule:
        dl_type: 0x800      # ipv4
        nw_proto: 17        # udp
        udp_dst: 53         # dns
        actions:
            output:
                set_fields:
```

(continues on next page)

(continued from previous page)

```

        - eth_dst: "72:b8:3c:4c:dc:4d"
        port: "s1" # s1 container
# Force DNS to our DNS server
- rule:
    dl_type: 0x800      # ipv4
    nw_proto: 6        # tcp
    tcp_dst: 53        # dns
    actions:
        output:
            set_fields:
                - eth_dst: "72:b8:3c:4c:dc:4d"
                port: "s1" # s1 container
- rule:
    actions:
        allow: 1        # allow

service-port-protect:
# Drop icmpv6 RAs
- rule:
    dl_type: 0x86dd    # ipv6
    nw_proto: 58       # icmpv6
    icmpv6_type: 134   # router advertisement
    actions:
        allow: 0        # drop
# Drop SMTP
- rule:
    dl_type: 0x800      # ipv4
    nw_proto: 6        # tcp
    tcp_dst: 25        # smtp
    actions:
        allow: 0        # drop
- rule:
    actions:
        allow: 1        # allow

```

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 41: `gauge.yaml`

```

# Recommended configuration is Prometheus for all monitoring, with all_dps: True
faucet_configs:
    - '/etc/faucet/faucet.yaml'
watchers:
    port_status_poller:
        type: 'port_state'
        all_dps: True
        #dps: ['sw1', 'sw2']
        db: 'prometheus'
    port_stats_poller:
        type: 'port_stats'
        all_dps: True
        #dps: ['sw1', 'sw2']
        interval: 10

```

(continues on next page)

(continued from previous page)

```
    db: 'prometheus'
    #db: 'influx'
  flow_table_poller:
    type: 'flow_table'
    all_dps: True
    interval: 60
    db: 'prometheus'
dbs:
  prometheus:
    type: 'prometheus'
    prometheus_addr: '0.0.0.0'
    prometheus_port: 9303
  ft_file:
    type: 'text'
    compress: True
    file: 'flow_table.yaml.gz'
  influx:
    type: 'influx'
    influx_db: 'faucet'
    influx_host: 'influxdb'
    influx_port: 8086
    influx_user: 'faucet'
    influx_pwd: 'faucet'
    influx_timeout: 10
```

1.5.1 Verifying configuration

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

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

1.5.2 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.

1.5.3 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
```

1.5.4 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/faucet/dps.yaml
  - /etc/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:
    ...
```

1.5.5 Configuration options

Top Level

Table 1: Faucet.yaml

Attribute	Type	Default	Description
acls	dictionary	{}	Configuration specific to acls. The keys are names of each acl, and the values are config dictionaries holding the acl's configuration (see below).
dps	dictionary	{}	Configuration specific to datapaths. The keys are names or dp_ids of each datapath, and the values are config dictionaries holding the datapath's configuration (see below).
meters	dictionary	{}	Configuration specific to meters. The keys are names of each meter, and the values are config dictionaries holding the meter's configuration (see below).
routers	dictionary	{}	Configuration specific to routers. The keys are names of each router, and the values are config dictionaries holding the router's configuration (see below).
version	integer	2	The config version. 2 is the only supported version.
vlans	dictionary	{}	Configuration specific to vlans. The keys are names or vids of each vlan, and the values are config dictionaries holding the vlan's configuration (see below).

DP

DP configuration is entered in the 'dps' configuration block. The 'dps' configuration contains a dictionary of configuration blocks each containing the configuration for one datapath. The keys can either be string names given to the

datapath, or the OFP datapath id.

Table 2: dps: <dp name or id>: {}

Attribute	Type	Default	Description
advertise_interval	type	30	How often to advertise (eg. IPv6 RAs)
arp_neighbor_timeout	type	250	ARP and neighbour timeout in seconds
description	string	name	Description of this datapath, strictly informational
dp_id	integer	The configuration key	the OFP datapath-id of this datapath
drop_bpdu	boolean	True	If True, Faucet will drop all STP BPDUs arriving at the datapath. NB: Faucet does not handle BPDUs itself, if you disable this then you either need to configure an ACL to catch BPDUs or Faucet will forward them as though they were normal traffic.
drop_broadcast_source_addresses	boolean	True	If True, Faucet will drop any packet from a broadcast source address
drop_lldp	boolean	True	If True, Faucet will drop all LLDP packets arriving at the datapath.
drop_spoofed_faucet_mac	boolean	True	If True, Faucet will drop any packet it receives with an ethernet source address equal to a MAC address that Faucet is using.
group_table	boolean	False	If True, Faucet will use the OpenFlow Group tables to flood packets. This is an experimental feature that is not fully supported by all devices and may not interoperate with all features of faucet.
group_table_routing	boolean	False	If True, Faucet will use the OpenFlow Group tables for routing (nexthops) This is an experimental feature that is not fully supported by all devices and may not interoperate with all features of faucet.
hardware	string	“Open vSwitch”	The hardware model of the datapath. Defaults to “Open vSwitch”. Other options can be seen in the documentation for valve.py
high_priority	integer	low_priority + 1 (9001)	The high priority value.
highest_priority	integer	high_priority + 98 (9099)	The highest priority number to use.
ignore_learn_ins	integer	10	Ignore every approx nth packet for learning. 2 will ignore 1 out of 2 packets; 3 will ignore 1 out of 3 packets. This limits control plane activity when learning new hosts rapidly. Flooding will still be done by the dataplane even with a packet is ignored for learning purposes.
interfaces	dictionary	{}	configuration block for interface specific config (see below)

Continued on next page

Table 2 – continued from previous page

Attribute	Type	Default	Description
interface_ranges	dictionary	{}	contains the config blocks for sets of multiple interfaces. The configuration entered here will be used as the defaults for these interfaces. The defaults can be overwritten by configuring the interfaces individually, which will also inherit all defaults not specifically configured. For example, if the range specifies tagged_vlans: [1, 2, 3], and the individual interface specifies tagged_vlans: [4], the result will be tagged_vlans: [4]. The format for the configuration key is a comma separated string. The elements can either be the name or number of an interface or a range of port numbers eg: "1-6,8,port9".
learn_ban_timeout	integer	10	When a host is rapidly moving between ports Faucet will stop learning mac addresses on one of the ports for this number of seconds.
learn_jitter	integer	10	In order to reduce load on the controller Faucet will randomly vary the timeout for learnt mac addresses by up to this number of seconds.
lldp_beacon	dict	{}	Configuration block for LLDP beacons
low_priority	integer	low_priority + 9000 (9000)	The low priority value.
lowest_priority	integer	priority_offset (0)	The lowest priority number to use.
max_host_fib_retry_count	integer	10	Limit the number of times Faucet will attempt to resolve a next-hop's l2 address.
max_hosts_per_resolve_cycle	integer	5	Limit the number of hosts resolved per cycle.
max_resolve_backoff_time	integer	32	When resolving next hop l2 addresses, Faucet will back off exponentially until it reaches this value.
metrics_rate_limit_sec	integer	0	Rate limit metric updates - don't update metrics if last update was less than this many seconds ago.
name	string	The configuration key	A name to reference the datapath by.
ofchannel_log	string	None	Name of logfile for openflow logs
packetin_pps	integer	None	Ask switch to rate limit packet pps.
priority_offset	integer	0	Shift all priority values by this number.
proactive_learn	boolean	True	whether proactive learning is enabled for IP nexthops
stack	dictionary	{}	configuration block for stacking config, for loop protection (see below)
timeout	integer	300	timeout for MAC address learning
use_idle_timeout	boolean	False	Turn on/off the use of idle timeout for src_table, default OFF.

Stacking (DP)

Stacking is configured in the dp configuration block and in the interface configuration block. At the dp level the following attributes can be configured withing the configuration block 'stack':

Table 3: dps: <dp name or id>: stack: { }

Attribute	Type	Default	Description
priority	integer	0	setting any value for stack priority indicates that this datapath should be the root for the stacking topology.

LLDP (DP)

LLDP beacons are configured in the dp and interface configuration blocks.

Note: the LLDP beacon service is specifically NOT to discover topology. It is intended to facilitate physical troubleshooting (e.g. a standard cable tester can display OF port information). A separate system will be used to probe link/neighbor activity, addressing issues such as authenticity of the probes.

The following attributes can be configured withing the 'lldp_beacon' configuration block at the dp level:

Table 4: dps: <dp name or id>: lldp_beacon: { }

Attribute	Type	Default	Description
system_name	string	The datapath name	seconds between sending beacons
send_interval	integer	None	seconds between sending beacons
max_per_interval	integer	None	the maximum number of beacons, across all ports to send each interval

Interfaces

Configuration for each interface is entered in the 'interfaces' configuration block withing the config for the datapath. Each interface configuration block is a dictionary keyed by the interface name.

Defaults for groups of interfaces can also be configured under the 'interface-ranges' attribute within the datapath configuration block. These provide default values for a number of interfaces which can be overwritten with the config block for an individual interface. These are keyed with a string containing a comma separated list of OFP port numbers, interface names or with OFP port number ranges (eg. 1-6).

Table 5: dps: <dp name or id>: interfaces: <interface name or OFP port number>: {}

Attribute	Type	Default	Description
acl_in	integer or string	None	Deprecated, replaced by acls_in which accepts a list. The acl that should be applied to all packets arriving on this port. referenced by name or list index
acls_in	a list of ACLs, as integers or strings	None	A list of ACLs that should be applied to all packets arriving on this port. referenced by name or list index. ACLs listed first take priority over those later in the list.
description	string	name (which defaults to the configuration key)	Description, purely informational
enabled	boolean	True	Allow packets to be forwarded through this port.
hairpin	boolean	False	If True it allows packets arriving on this port to be output to this port. This is necessary to allow routing between two vlans on this port, or for use with a WIFI radio port.
lldp_beacon	dict	{}	Configuration block for lldp configuration
loop_protect	boolean	False	If True, do simple loop protection on this port.
max_hosts	integer	255	the maximum number of mac addresses that can be learnt on this port.
mirror	a list of integers or strings	None	Mirror all packets recieved and transmitted on the ports specified (by name or by port number), to this port.
name	string	The configuration key.	a name to reference this port by.
native_vlan	integer	None	The vlan associated with untagged packets arriving and leaving this interface.
number	integer	The configuration key.	The OFP port number for this port.
opstatus_reconf	boolean	True	If True, FAUCET will reconfigure the pipeline based on operational status of the port.
output_only	boolean	False	If True, no packets will be accepted from this port.
override_output_port	integer	None	If set, packets are sent to this other port.
permanent_learn	boolean	False	When True Faucet will only learn the first MAC address on this interface. All packets with an ethernet src address not equal to that MAC address will be dropped.
stack	dictionary	None	configuration block for interface level stacking configuration
tagged_vlans	list of integers or strings	None	The vlans associated with tagged packets arriving and leaving this interfaces.
unicast_flood	boolean	True	If False unicast packets will not be flooded to this port.

Stacking (Interfaces)

Stacking port configuration indicates how datapaths are connected when using stacking. The configuration is found under the 'stack' attribute of an interface configuration block. The following attributes can be configured:

Table 6: dps: <dp name or id>; interfaces: <interface name or port number>; stack: { }

Attribute	Type	Default	Description
dp	integer or string	None	the name of dp_id of the dp connected to this port
port	integer or string	None	the name or OFP port number of the interface on the remote dp connected to this interface.

LLDP (Interfaces)

Interface specific configuration for LLDP.

Table 7: dps: <dp name or id>; interfaces: <interface name or port number>; lldp_beacon: { }

Attribute	Type	Default	Description
enable	boolean	False	Enable sending lldp beacons from this interface
org_tlvs	list	[]	Definitions of Organisational TLVs to add to LLDP beacons
port_descr	string	Interface description	Port description to use in beacons from this interface
system_name	string	lldp_beacon (dp) system name	The System Name to use in beacons from this interface

LLDP Organisational TLVs (Interfaces)

Faucet allows defining organisational TLVs for LLDP beacons. These are configured in a list under lldp_beacons/org_tlvs at the interfaces level of configuration.

Each list element contains a dictionary with the following elements:

Table 8: dps: <dp name or id>; interfaces: <interface name or port number>; lldp_beacon: org_tlvs: - { }

Attribute	Type	Default	Description
info	string	None	the info field of the tlv, as a hex string
oui	integer	None	the Organisationally Unique Identifier
subtype	integer	None	The organizationally defined subtype

Router

Routers config is used to allow routing between vlans. Routers configuration is entered in the 'routers' configuration block at the top level of the faucet configuration file. Configuration for each router is an entry in the routers dictionary and is keyed by a name for the router. The following attributes can be configured:

Table 9: routers: <router name>: { }

Attribute	Type	Default	Description
vlan	list of integers or strings	None	Enables inter-vlan routing on the given vlans

VLAN

VLANs are configured in the ‘vlans’ configuration block at the top level of the faucet config file. The config for each vlan is an entry keyed by its vid or a name. The following attributes can be configured:

Table 10: vlans: <vlan name or vid>: {}

Attribute	Type	Default	Description
acl_in	string or integer	None	Deprecated, replaced by acls_in which accepts a list. The acl to be applied to all packets arriving on this vlan.
acls_in	a list of ACLs, as integers or strings	None	The acl to be applied to all packets arriving on this vlan. ACLs listed first take priority over those later in the list.
bgp_as	integer	None	The local AS number to used when speaking BGP
bgp_connect_mode	string	“both”	Whether to try to connect to natives (“active”), listen only (“passive”), or “both”.
bgp_local_address	string (IP Address)	None	The local address to use when speaking BGP
bgp_neighbour_addresses	list of strings (IP Addresses)	None	The list of BGP neighbours
bgp_neighbour_as	integer	None	The AS Number for the BGP neighbours
bgp_port	integer	9179	Port to use for bgp sessions
description	string	None	Strictly informational
faucet_vips	list of strings (IP address prefixes)	None	The IP Address for Faucet’s routing interface on this vlan
max_hosts	integer	255	The maximum number of hosts that can be learnt on this vlan.
minimum_ip_size_check	boolean	True	If False, don’t check that IP packets have a payload (must be False for OVS trace/tutorial to work)
name	string	the configuration key	A name that can be used to refer to this vlan.
proactive_arp_limit	integer	2052	Do not proactively ARP for hosts once this value has been reached (set to None for unlimited)
proactive_nd_limit	integer	2052	Don’t proactively discover IPv6 hosts once this value has been reached (set to None for unlimited)
routes	list of routes	None	static routes configured on this vlan (see below)
targeted_gw_resolution	boolean	False	if True, and a gateway has been resolved, target the first re-resolution attempt to the same port rather than flooding.
unicast_flood	boolean	True	If False packets to unknown ethernet destination MAC addresses will be dropped rather than flooded.
vid	integer	the configuration key	The vid for the vlan.

Static Routes

Static routes are given as a list. Each entry in the list contains a dictionary keyed with the keyword ‘route’ and contains a dictionary configuration block as follows:

Table 11: vlans: <vlan name or vid>: routes: - route: { }

Attribute	Type	Default	Description
ip_dst	string (IP subnet)	None	The destination subnet.
ip_gw	string (IP address)	None	The next hop for this route

Meters

Note: Meters are platform dependent and not all functions may be available.

Meters are configured under the ‘meters’ configuration block. The meters block contains a dictionary of individual meters each keyed by its name.

Table 12: meters: <meter name>:

Attribute	Type	Default	Description
meter_id	int		Unique identifier.
entry	dict		Defines the meter actions. Details Below.

Table 13: : meters: <meter name>: entry:

Attribute	Type	Default	Description
flags	String or list of String	KBPS	Possible values are ‘KBPS’ (Rate value in kb/s (kilo-bit per second).), ‘PKTPS’ (Rate value in packet/sec.), ‘BURST’ (Do burst size), ‘STATS’ (Collect statistics)
bands	list of bands (which are dicts, see below)		

Table 14: : meters: <meter name>: entry: bands:

Attribute	Type	Default	Description
type	String		‘DROP’ - drop packets when the band rate is exceeded, or ‘DSCP_REMARK’ - use a simple DiffServ policer to remark the DSCP field in the IP header of packets that exceed the band rate.
rate	int		Rate for dropping or remarking packets, depending on the above type. Value is in KBPS or PKTPS flag depending on the flag set.
burst_size	int		Only used if flags includes BURST. Indicates the length of packet or byte burst to consider for applying the meter.
prec_level	int		Only used if type is DSCP_REMARK. The amount by which the drop precedence should be increased.

ACLs

ACLs are configured under the ‘acls’ configuration block. The acls block contains a dictionary of individual acls each keyed by its name.

Each acl contains a list of rules: a packet will have the first matching rule applied to it.

Each rule is a dictionary containing the single key ‘rule’ with matches and actions. Matches are key/values based on the ryu RESTful API. Actions is a dictionary of actions to apply upon match.

Table 15: : acls: <acl name>: - rule: actions: { }

Attribute	Type	Default	Description
allow	boolean	False	If True allow the packet to continue through the Faucet pipeline, if False drop the packet.
force_port_vlan	boolean	False	if True, don’t verify VLAN/port association.
cookie	int, 0-2**16	defaults to datapath cookie value	If set, cookie on this flow will be set to this value.
meter	string	None	meter to apply to the packet
mirror	string or integer	None	Copy the packet, before any modifications, to the specified port (NOTE: mirroring is done in input direction only)
output	dict	None	used to output a packet directly. Details below.

The output action contains a dictionary with the following elements:

Table 16: : acls: <acl name>: - rule: actions: output: { }

Attribute	Type	Default	Description
set_fields	list of dicts	None	A list of fields to set with values, eg. eth_dst: “1:2:3:4:5:6”
port	integer or string	None	The port to output the packet to.
ports	list of [integer or string]	None	The list of ports the packet will be output through.
pop_vlans	boolean	False	Pop vlan tag before output.
vlan_vid	integer	False	Push vlan tag before output.
swap_vid	integer	None	Rewrite the vlan vid of the packet when outputting
vlan_vids	list of [integer or { integer, eth_type }]	None	Push vlan tags on output, with optional eth_type.
failover	dict	None	Output with a failover port (see below).

Failover is an experimental option, but can be configured as follows:

Table 17: : acls: <acl name>: - rule: actions: output: failover: { }

Attribute	Type	Default	Description
group_id	integer	None	The OFP group id to use for the failover group
ports	list	None	The list of ports the packet can be output through.

1.5.6 Environment variables

You can use environment variables to override default behaviour of faucet such as paths for configuration files and port numbers.

Environment Variable	Type	Default	Description
FAUCET_CONFIG	Colon-separated list of file paths	/etc/faucet/faucet.conf	Location for faucet configuration file, can be special values STDOUT or STDERR
FAUCET_CONFIG_STAT_RELOAD	Boolean	False	If true, faucet will automatically reload itself and apply new configuration when FAUCET_CONFIG changes
FAUCET_LOG_LEVEL	Python log level	INFO	Log verbosity
FAUCET_LOG	File path or STDOUT or STDERR	/var/log/faucet/faucet.log	Location for faucet to log messages to, can be special values STDOUT or STDERR
FAUCET_EXCEPTION_LOG	File path or STDOUT or STDERR	/var/log/faucet/faucet.log	Location for faucet to log exceptions to, can be special values STDOUT or STDERR
FAUCET_EVENT SOCK	Socket path		Location to a UNIX socket where faucet will write events to, or empty to disable events
FAUCET_PROMETHEUS_PORT	Port	9302	TCP port to listen on for faucet prometheus client
FAUCET_PROMETHEUS_ADDR	IP Address	0.0.0.0	IP address to listen on for faucet prometheus client
FAUCET_PIPELINE_DIR	Colon-separated list of file paths	/etc/faucet:etc/faucet	Location for faucet to load pipeline definitions from the first valid directory in list
GAUGE_CONFIG	Colon-separated list of file paths	/etc/faucet/gauge.conf	Location for gauge configuration file, can be special values STDOUT or STDERR
GAUGE_CONFIG_STAT_RELOAD	Boolean	False	If true, gauge will automatically reload itself and apply new configuration when GAUGE_CONFIG changes
GAUGE_LOG_LEVEL	Python log level	INFO	Log verbosity
GAUGE_LOG	File path or STDOUT or STDERR	/var/log/faucet/gauge.log	Location for gauge to log messages to, can be special values STDOUT or STDERR
GAUGE_EXCEPTION_LOG	File path or STDOUT or STDERR	/var/log/faucet/gauge.log	Location for gauge to log exceptions to, can be special values STDOUT or STDERR
GAUGE_PROMETHEUS_ADDR	IP Address	0.0.0.0	IP address to listen on for gauge prometheus client

1.6 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.

1.6.1 Forwarding

1.6.2 Routing

1.6.3 Policy

1.7 Vendor-specific Documentation

1.7.1 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](#).

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
```

(continues on next page)

(continued from previous page)

```

/* (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

```

(continues on next page)

(continued from previous page)

```
* 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/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.

Listing 42: /etc/faucet/faucet.yaml

```
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](#)

1.7.2 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

System & Network Requirements

- Use Serial Console cable to login to the box.

- Use `minicom` for serial terminal @ 115Kbps. Minicom is available on Linux and MacOS (macports) systems.
- Connected Port 1 of Switch to Top of the Rack (TOR) switch which had DHCP and DNS enabled. Mac Address was programmed into DNS/DHCP Server so that IP address of 10.20.5.11 was provided to this box.
- Need a TFTP Server on the network with write access so that we can store system software for upgrade and also certificates. The switch can copy files from a TFTP Server.

Tip: How to clear the password settings

Simultaneously press “Reset” and “Clear” buttons using a paper clip. Release “Reset” button only first. Once the orange power light comes up (after ~5 seconds), release the “Clear” button.

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

// If you have mixed both management and control-plane vlan to a single port (port 1)
switch (config)# vlan 2048 untagged 1

// Alternatively, you can have a separate port for control plane traffic
// 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
// May Not be needed if you have port 1 set to DHCP/Bootp/DNS IP address of 10.20.5.11
```

(continues on next page)

(continued from previous page)

```

switch (config)# vlan 2048 ip address 10.20.5.11/16

// Alternatively, to configure only 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

// Configure DNS. Here DNS is set to a local LAN DNS server
switch (config)# ip dns server-address priority 1 10.20.0.1

```

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

// Faucet Controller name can be FQDN
switch(openflow)# controller-id 1 hostname controller-1.tenant1.tenants.
↳servicefractal.com 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

// Gauge Controller name can be FQDN
switch(openflow)# controller-id 2 hostname controller-1.tenant1.tenants.
↳servicefractal.com 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

// Faucet Controller name can be FQDN
switch(openflow)# controller-id 1 hostname controller-1.tenant1.tenants.
↳servicefractal.com 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

// Gauge Controller name can be FQDN
switch(openflow)# controller-id 2 hostname controller-1.tenant1.tenants.
↳servicefractal.com 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

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

// 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

// To save the Configuration
switch# save
switch# write mem

// Show running Configuration
switch# show running-config

// 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
```

(continues on next page)

(continued from previous page)

```
switch# show openflow instance aggregate | include Datapath ID
          Datapath ID                : 00013863bbc41800
```

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 ID	Table Name	Flow Count	Miss 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.

```
switch# show openflow instance aggregate
```

```
    Configured OF Version      : 1.3 only
    Negotiated OF Version      : 1.3
    Instance Name              : aggregate
    Data-path Description       : aggregate
    Administrator Status        : Enabled
    Member List                 : VLAN 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
```

```
→11, 12,
```

```
    .....
    .....
```

Controller Id	Connection Status	Connection State	Secure	Role
1	Connected	Active	Yes	Equal
2	Connected	Active	Yes	Equal

```
// To just get openflow controllers
```

(continues on next page)

(continued from previous page)

```

switch (openflow)# show openflow controllers

                Controller Information

                Controller Id IP Address      Hostname          Port   Interface
                -----
↪-
                1             0.0.0.0        controller-1.t... 6653   VLAN 2048
                2             0.0.0.0        controller-1.t... 6654   VLAN 2048

// Copy Running Config to a TFTP Server
// (first enable TFTP client)
switch (config)# tftp client

```

PKI Setup on switch

Note: Root certificate container supports only one root certificate not a chain. So, install the one that the CSR (Certificate Signing Request) is signed with.

```

switch# show crypto pki application

    Certificate Extension Validation :

    Application      SAN/CN
    -----
    openflow         Disabled
    syslog           Disabled

// Here, we create Service Fractal CA profile
switch (config)# crypto pki ta-profile SERVICEFRACTAL_CA

// Copy the root certificate for the SERVICEFRACTAL_CA from a tftp server
switch# copy tftp ta-certificate SERVICEFRACTAL_CA 10.10.22.15 tenant1.tenants.
↪servicefractal.com.cert.pem

switch# show crypto pki ta-profile SERVICEFRACTAL_CA
Profile Name      Profile Status          CRL Configured  OCSP Configured
-----
SERVICEFRACTAL_CA 1 certificate installed      No              No

Trust Anchor:
Version: 3 (0x2)
Serial Number: 4096 (0x1000)
Signature Algorithm: sha256withRSAEncryption
...
.....

// Now we are ready to create a CSR so that a switch identity certificate_
↪that is accepted by the controller can be setup.

switch (config)# crypto pki identity-profile hpe_sf_switch1 subject common-name_
↪myswitch.tenant1.tenants.servicefractal.com org ServiceFractal org-unit vendor-test_
↪locality MyCity state CA country US

```

(continues on next page)

(continued from previous page)

```

switch (config)# show crypto pki identity-profile
  Switch Identity:
    ID Profile Name      : hpe_sf_switch1
    Common Name (CN)    : myswitch.tenant1.tenants.servicefractal.com
    Org Unit (OU)       : vendor-test
    Org Name (O)        : ServiceFractal
    Locality (L)        : MyCity
    State (ST)          : CA
    Country (C)         : US

// Generate CSR
switch (config)# crypto pki create-csr certificate-name hpeswt_switch1_crt ta-profile_
↪SERVICEFRACTAL_CA usage openflow

// Copy the printed CSR request and send it to "SERVICEFRACTAL_CA"

switch (config)# show crypto pki local-certificate summary
  Name                Usage          Expiration          Parent / Profile
  -----
  hpeswt_switch1_crt  Openflow          CSR                  SERVICEFRACTAL_CA

// Once the signed certificate is received, copy the same to switch.
switch (config)# copy tftp local-certificate 10.10.22.15 myswitch.tenant1.tenants.
↪servicefractal.com.cert.pem
  000M Transfer is successful

switch (config)# show crypto pki local-certificate summary
  Name                Usage          Expiration          Parent / Profile
  -----
  hpeswt_switch1_crt  Openflow          2019/01/02          SERVICEFRACTAL_CA

```

Faucet

On the FAUCET configuration file (`/etc/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.

Listing 43: `/etc/faucet/faucet.yaml`

```

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

(continues on next page)

(continued from previous page)

```
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 OS version as of Dec 2017 is 16.05
- Aruba Switches
- FAUCET
- Model 2390F Product Site
- 2930F top level documentation
- Password settings
- PKI Setup

1.7.3 Faucet on Lagopus

Introduction

Lagopus 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 according to the [quickstart guide](#). 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).

Listing 44: /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

Listing 45: /etc/faucet/faucet.yaml

```
vlan:
  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

```
faucet --verbose --ryu-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:

Listing 46: /var/log/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

```

1.7.4 Faucet on ZodiacFX

Introduction

ZodiacFX is a small 4 port multi table OF1.3 switch from Northbound Networks.

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 47: 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+]]

```

(continues on next page)

(continued from previous page)

```

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:

```

$ sudo ./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.

```

1.7.5 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 faucet controllerid 1 priority 1 ipaddr 10.0.1.
↪8 port 6653 security none
set config controller controllergroup gauge controllerid 1 priority 1 ipaddr 10.0.1.8
↪port 6654 security none
set config switch dpid 0x1
```

Configure the tables

These matches are known to pass the unit tests as of FAUCET 1.6.18, but take care to adjust ACL tables matches based on the type of ACL rules defined in the configuration file. Different FAUCET releases may also use different match fields in the other tables.

```
set config pipeline tablesizes 1524 1024 1024 5000 3000 1024 1024 5000 1024
↪tablewidths 80 40 40 40 40 40 40 40 40
set config table tableid 0 matchfields 0 3 4 5 6 10 11 12 13 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
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 29
set config table tableid 7 matchfields 0 3 6
set config table tableid 8 matchfields 0 3 6
```

Note that this table configuration will allow most of the automated test cases to pass, except FaucetIPv6TupleTest (which requires IPv6 Src and Dst matching in the ACL table). In order to run this test, table 0 must be configured as follows:

```
set config table tableid 0 matchfields 0 5 6 10 26 27 13 14
```

Create faucet.yaml

Listing 48: /etc/faucet/faucet.yaml

```
vlan:
  100:
    name: "test"
dps:
  noviflow-1:
    dp_id: 0x1
    hardware: "NoviFlow"
```

(continues on next page)

(continued from previous page)

```

interfaces:
  1:
    native_vlan: 100
  2:
    native_vlan: 100

```

Run 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:

Listing 49: /var/log/faucet/faucet.log

```

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

```

1.7.6 Faucet on OVS with DPDK

Introduction

Open vSwitch 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 [official OVS instructions](#), including enabling DPDK at compile time and in OVS's initial configuration.

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

```
$ sudo /usr/local/share/openvswitch/scripts/ovs-ctl stop
* Exiting ovs-vswitchd (20510)
* Exiting ovssdb-server (20496)
$ sudo /usr/local/share/openvswitch/scripts/ovs-ctl start
```

(continues on next page)

(continued from previous page)

```

* 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 enpls0f0 type=dpdk options:dpdk-
↳devargs=0000:01:00.0
ovs-vsctl add-port br0 dpdk1 -- set interface enpls0f1 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-ofctl show br0
ovs-vsctl get bridge br0 datapath_id

```

Create faucet.yaml

Note: Change `dp_id`, to the value reported above, prefaced with “0x”.

Listing 50: `/etc/faucet/faucet.yaml`

```

vlans:
  100:
    name: "test"
dps:

```

(continues on next page)

(continued from previous page)

```
ovsdpdk-1:
  dp_id: 0x000090e2ba7e7564
  hardware: "Open vSwitch"
  interfaces:
    1:
      native_vlan: 100
    2:
      native_vlan: 100
```

Run FAUCET

```
faucet --verbose --ryu-ofp-listen-host=127.0.0.1
```

Test connectivity

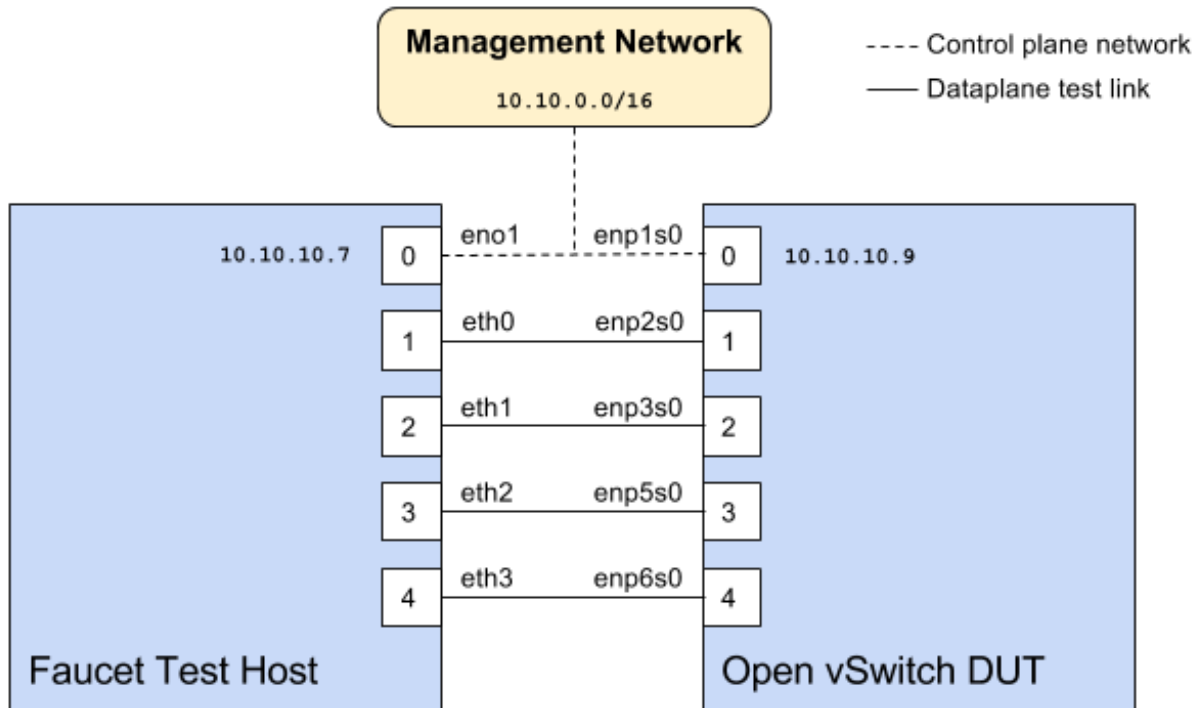
Host(s) on enp1s0f0 and enp1s0f1 in the same IP subnet, should now be able to communicate, and FAUCET's log file should indicate learning is occurring:

Listing 51: /var/log/faucet/faucet.log

```
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
```

1.7.7 Faucet Testing with OVS on Hardware

Setup



Faucet configuration file

Listing 52: /etc/faucet/hw_switch_config.yaml

```
# Faucet Configuration file: /etc/faucet/hw_switch_config.yaml
#
# If hw_switch value set to True, map a hardware OpenFlow switch to ports on this_
↪ machine.
# Otherwise, run tests against OVS locally.
hw_switch: True
hardware: 'Open vSwitch'
dp_ports:
  1: eth0
  2: eth1
  3: eth2
  4: eth3

# Hardware switch's DPID
dpid: 0xacd28f18b
cpn_intf: eno1
of_port: 6636
gauge_of_port: 6637
```

Hardware

1. For Network Interface Cards (NICs), prefer Intel branded models.
2. I have also used [Hi-Speed USB to dual Ethernet](#) which works great

Software

1. Ubuntu 16.04 Xenial
2. Open vSwitch 2.7.2+

Commands

Commands to be executed on each side - **Faucet Test host** and **Open vSwitch**.

Commands on Faucet Test Host

Run these commands as root on the Ubuntu system (v16.04 used)

```
$ sudo mkdir -p /usr/local/src/
$ sudo mkdir -p /etc/faucet/
$ sudo cd /usr/local/src/
$ sudo git clone https://github.com/faucetsdn/faucet.git
$ cd faucet
$ sudo ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default_
↪qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default_
↪qlen 1000
link/ether b4:96:91:00:88:a4 brd ff:ff:ff:ff:ff:ff
inet6 fe80::b696:91ff:fe00:88a4/64 scope link
valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default_
↪qlen 1000
link/ether b4:96:91:00:88:a5 brd ff:ff:ff:ff:ff:ff
inet6 fe80::b696:91ff:fe00:88a5/64 scope link
valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default_
↪qlen 1000
link/ether b4:96:91:00:88:a6 brd ff:ff:ff:ff:ff:ff
inet6 fe80::b696:91ff:fe00:88a6/64 scope link
valid_lft forever preferred_lft forever
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default_
↪qlen 1000
link/ether b4:96:91:00:88:a7 brd ff:ff:ff:ff:ff:ff
inet6 fe80::b696:91ff:fe00:88a7/64 scope link
valid_lft forever preferred_lft forever
6: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default_
↪qlen 1000
```

(continues on next page)

(continued from previous page)

```
link/ether 00:1e:67:ff:f6:80 brd ff:ff:ff:ff:ff:ff
inet 10.10.10.7/16 brd 10.20.255.255 scope global eno1
valid_lft forever preferred_lft forever
inet6 cafe:babe::21e:67ff:feff:f680/64 scope global mngtmpaddr dynamic
valid_lft 86398sec preferred_lft 14398sec
inet6 fe80::21e:67ff:feff:f680/64 scope link
valid_lft forever preferred_lft forever
```

Tip: To locate the corresponding physical port, you can make the port LED blink with *Ethtool*.

Commands on Open vSwitch

Login as root on the Ubuntu system and install OpenvSwitch and start openvswitch-switch service

```
$ sudo apt-get install openvswitch-switch
$ sudo systemctl status openvswitch-switch.service
$ sudo ovs-vsctl add-br ovs-br0
$ sudo ovs-vsctl add-port ovs-br0 enp2s0 -- set Interface enp2s0 ofport_request=1
$ sudo ovs-vsctl add-port ovs-br0 enp3s0 -- set Interface enp3s0 ofport_request=2
$ sudo ovs-vsctl add-port ovs-br0 enp5s0 -- set Interface enp5s0 ofport_request=3
$ sudo ovs-vsctl add-port ovs-br0 enp6s0 -- set Interface enp6s0 ofport_request=4
$ sudo ovs-vsctl set-fail-mode ovs-br0 secure
$ sudo ovs-vsctl set bridge ovs-br0 protocols=OpenFlow13
$ sudo ovs-vsctl set-controller ovs-br0 tcp:10.10.10.7:6636 tcp:10.10.10.7:6637
$ sudo ovs-vsctl get bridge ovs-br0 datapath_id
$ sudo ovs-vsctl show
308038ec-495d-412d-9b13-fe95bda4e176
    Bridge "ovs-br0"
        Controller "tcp:10.10.10.7:6636"
        Controller "tcp:10.10.10.7:6637"
        Port "enp3s0"
            Interface "enp3s0"
        Port "enp2s0"
            Interface "enp2s0"
        Port "enp6s0"
            Interface "enp6s0"
        Port "ovs-br0"
            Interface "ovs-br0"
                type: internal
        Port "enp5s0"
            Interface "enp5s0"
                type: system
    ovs_version: "2.7.0"

$ sudo ovs-vsctl -- --columns=name,ofport list Interface
name                : "ovs-br0"
ofport              : 65534

name                : "enp5s0"
ofport              : 3

name                : "enp2s0"
ofport              : 1
```

(continues on next page)

(continued from previous page)

```

name          : "enp6s0"
ofport        : 4

name          : "enp3s0"
ofport        : 2

```

Tip: To locate the corresponding physical port, you can make the port LED blink with *Ethtool*.

Check port speed information to make sure that they are at least 1Gbps

```

$ sudo ovs-ofctl -O OpenFlow13 dump-ports-desc ovs-br0
OFPST_PORT_DESC reply (OF1.3) (xid=0x2):
  1(enp2s0): addr:00:0e:c4:ce:77:25
    config:      0
    state:       0
    current:     1GB-FD COPPER AUTO_NEG
    advertised:  10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    speed: 1000 Mbps now, 1000 Mbps max
  2(enp3s0): addr:00:0e:c4:ce:77:26
    config:      0
    state:       0
    current:     1GB-FD COPPER AUTO_NEG
    advertised:  10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    speed: 1000 Mbps now, 1000 Mbps max
  3(enp5s0): addr:00:0e:c4:ce:77:27
    config:      0
    state:       0
    current:     1GB-FD COPPER AUTO_NEG
    advertised:  10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-FD COPPER AUTO_NEG AUTO_
↔PAUSE
    speed: 1000 Mbps now, 1000 Mbps max
  4(enp6s0): addr:00:0a:cd:28:f1:8b
    config:      0
    state:       0
    current:     1GB-FD COPPER AUTO_NEG
    advertised:  10MB-HD COPPER AUTO_NEG AUTO_PAUSE AUTO_PAUSE_ASYM
    supported:   10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-HD 1GB-FD COPPER AUTO_NEG
    speed: 1000 Mbps now, 1000 Mbps max
LOCAL(ovs-br0): addr:00:0a:cd:28:f1:8b
  config:      PORT_DOWN
  state:       LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max

```

Running the tests

Edit the `/etc/faucet/hw_switch_config.yaml` file as shown earlier in this document setting `hw_switch=False` initially for testing.

```
$ sudo cp /usr/local/src/faucet/hw_switch_config.yaml /etc/faucet/hw_switch_config.  
→yaml  
$ sudo $EDITOR /etc/faucet/hw_switch_config.yaml  
$ cd /usr/local/src/faucet/
```

Install docker by following the *Installing docker* section and then run the hardware based tests by following the *Running the tests* section.

Once the above minitest version is successful with `hw_switch=False`, then edit the `/etc/faucet/hw_switch_config.yaml` file and set `hw_switch=True`.

Run tests again, verify they all pass.

Debugging

TCPDump

Many times, we want to know what is coming in on a port. To check on interface `enp2s0`, for example, use

```
$ sudo tcpdump -A -w enp2s0_all.pcap -i enp2s0
```

Or

```
$ sudo tcpdump -A -w enp2s0_all.pcap -i enp2s0 'dst host <controller-ip-address> and_  
→port 6653'
```

To read the pcap file, use

```
$ sudo tcpdump -r enp2s0_all.pcap
```

More detailed examples are available @ https://www.wains.be/pub/networking/tcpdump_advanced_filters.txt

Note: On which machine should one run tcpdump?

Depends, if you want to examine the `packet_ins` that are sent from switch to controller, run on the switch listening on the interface that is talking to the controller. If you are interested on what is coming in on a particular test port, then run it on the Test Host on that interface.

Ethtool

To locate a physical port say `enp2s0`, make the LED blink for 5 seconds:

```
$ sudo ethtool -p enp2s0 5
```

To figure out speed on the interface. Note that if Speed on the interface is at least not 1G, then tests may not run correctly.

```
$ sudo ethtool enp2s0  
$ sudo ethtool enp2s0 | grep Speed
```


References

<https://www.garron.me/en/linux/ubuntu-network-speed-duplex-lan.html>

1.8 External Resources

1.8.1 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/>
- <https://blog.cyberreboot.org/building-a-software-defined-network-with-raspberry-pis-and-a-zodiac-fx-switch-97184032cdc1>

1.8.2 Tutorial Videos

- <https://www.youtube.com/watch?v=fuqzzjmcwII>

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 and integration tests must pass (please use the docker based tests; see *Software switch testing with docker*).
- You must add a test if FAUCET's functionality changes (ie. a new feature, or correcting a bug).
- Please use the supplied git pre-commit hook (see `../git-hook/pre-commit`), to automatically run the unit tests and pylint for you at git commit time.
- Please enable TravisCI testing on your repo, which enables the maintainers to quickly verify that your changes pass all tests in a pristine environment.
- 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 Faucet Development Environment

A common way of developing faucet is inside a `virtualenv` with an IDE such as `PyCharm`.

Instructions on setting up PyCharm for developing faucet are as follows:

Create a new project in PyCharm

Set the `Location` of the project to the directory where a checked out copy of the faucet code from git is, for this tutorial I will assume the path is `/Dev/faucet/`.

Ignore the `Project Interpreter` settings for now, we will set those up after the project is created.

Click `Create` when you have completed these steps.

When asked `Would you like to create a project from existing sources instead?` click `Yes`.

Create virtual environment

Now that the project is created and source code imported, click the `File -> Settings` menu. In the dialog box that opens click the `Project: faucet -> Project Interpreter` sub menu.

Click the cog and select `Add...`

Under `Virtualenv Environment` you want to select `New environment` and select a `Location` for the virtualenv (which can be inside the directory where the faucet code lives, e.g `/Dev/faucet/venv`).

The `Base interpreter` should be set to `/usr/bin/python3`.

Click `Ok` which will create the virtualenv.

Now while that virtualenv builds and we still have the settings dialog open we will tweak a few project settings to make them compatible with our code style. Click on the `Tools -> Python Integrated Tools` menu and change the `Docstring format` to `Google`.

Finally, click `Ok` again to get back to the main screen of PyCharm.

Install requirements

Inside the PyCharm editor window we should now get a bar at the top of the window telling us of missing package requirements, click the `Install requirements` option to install the dependencies for faucet.

Create log and configuration directories

Now we need to create a log and configuration directory so that faucet can start:

```
mkdir -p /Dev/faucet/venv/var/log/faucet/
mkdir -p /Dev/faucet/venv/etc/faucet/
```

Copy the sample faucet configuration file from `/Dev/faucet/etc/faucet/faucet.yaml` to `/Dev/faucet/venv/etc/faucet/` and edit this configuration file as necessary.

Copy the sample gauge configuration file from `/Dev/faucet/etc/faucet/gauge.yaml` to `/Dev/faucet/venv/etc/faucet/` and edit this configuration file as necessary.

Configure PyCharm to run faucet

Now we need to configure PyCharm to run faucet, gauge and the unit tests.

First, click the `Run -> Run...` menu, then select the `Edit Configurations...` option to get to the build settings dialog.

We will edit the default `faucet` run configuration that has been created for us. First change the `Script` path to point to `ryu-manager` inside the virtualenv, for me this was `../venv/bin/ryu-manager`. Then set the `Parameters` to `faucet.faucet`. Make sure the working directory is set to `/Dev/faucet/faucet/`.

We will also add a `gauge` run configuration for starting `gauge`. First change the `Script` path to point to `ryu-manager` inside the virtualenv, for me this was `../venv/bin/ryu-manager`. Then set the `Parameters` to `faucet.gauge`. Make sure the working directory is set to `/Dev/faucet/faucet/`.

For running tests we need a few additional dependencies installed, I couldn't work out how to do this through PyCharm so run this command from a terminal window to install the correct dependencies inside the virtualenv:

```
/Dev/faucet/venv/bin/pip3 install -r /Dev/faucet/test-requirements.txt
```

Click the green plus icon to add a new build configuration, select `Python tests -> Unittests`. You can provide a `Name of Faucet Unit Tests` for the run configuration. For `Target` select `Script` path and enter the path `/Dev/faucet/tests`. For `Pattern` enter `test_*.py`.

You can click `Apply` and `Close` now that we've added all our new run configuration.

Now that everything is setup you can run either the `faucet` controller, `gauge` controller and test suite from the `Run` menu.

2.1.4 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.
- **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.

2.1.5 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

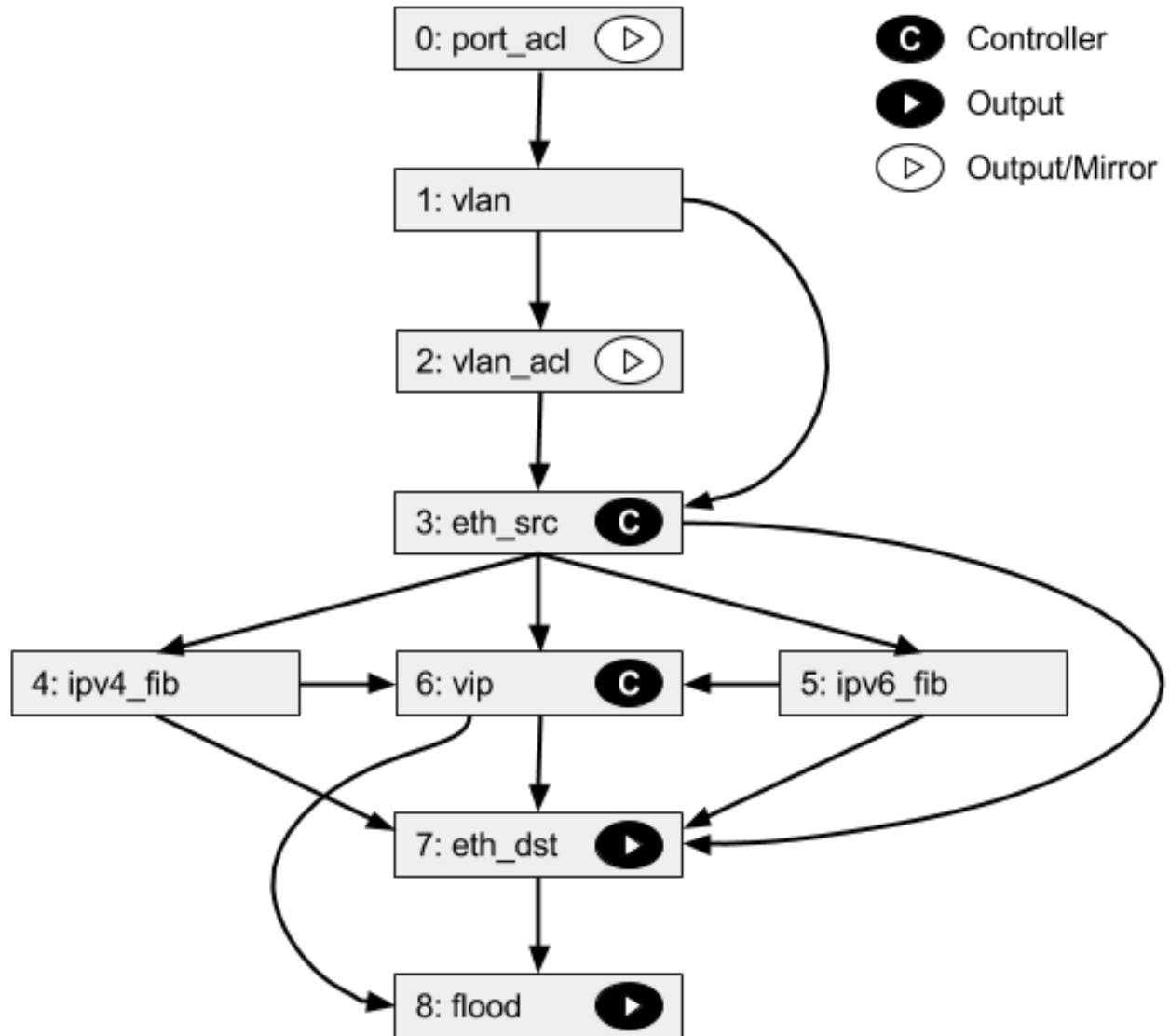


Table 0: PORT_ACL

- 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)

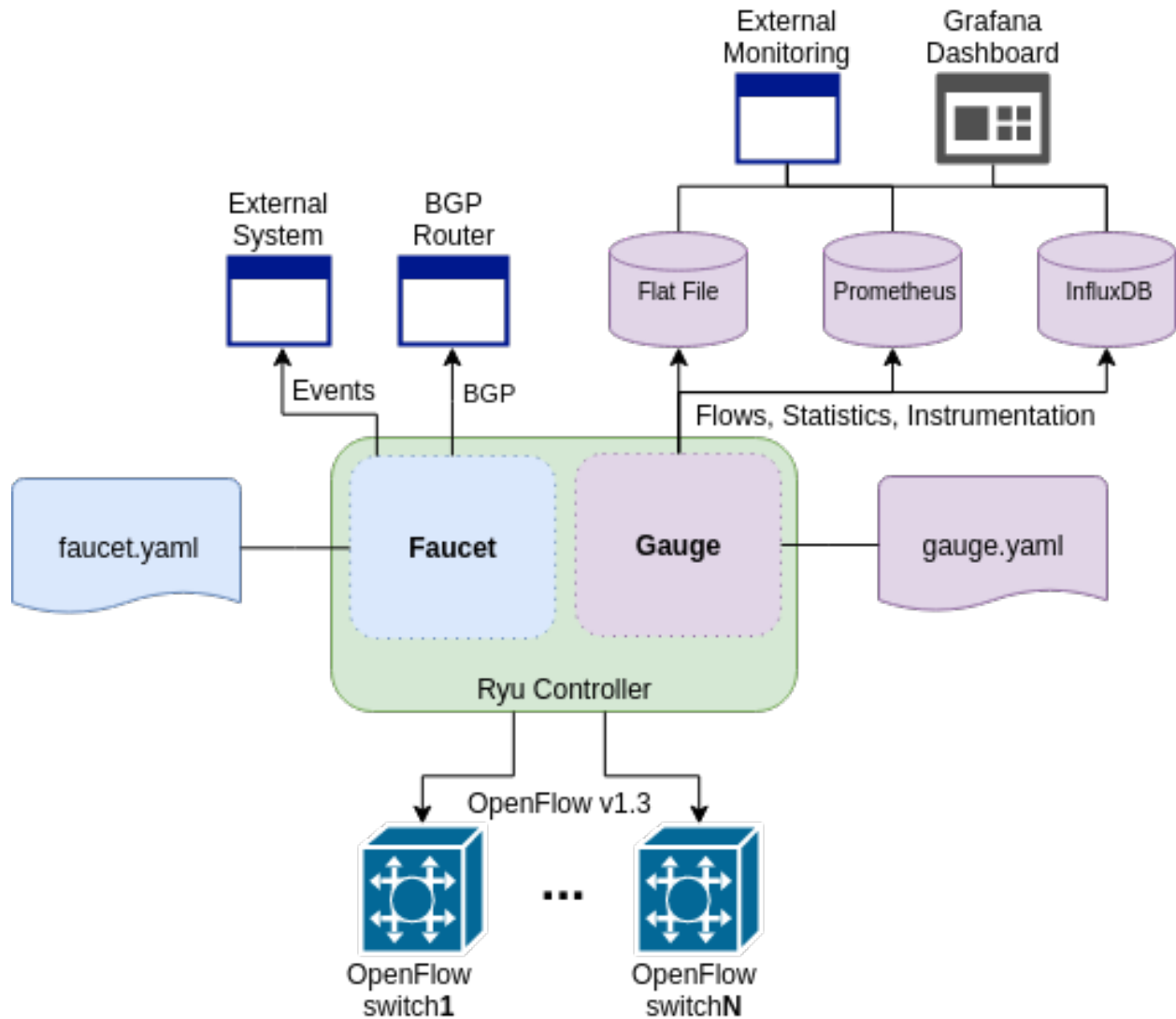
Table 7: ETH_DST

- 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 Installing docker

First, get yourself setup with docker based on our [Installing docker](#) documentation.

2.3.2 Software switch testing with docker

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

```
sudo docker build --pull -t faucet/tests -f Dockerfile.tests .
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.tcpdump
```

(continues on next page)

(continued from previous page)

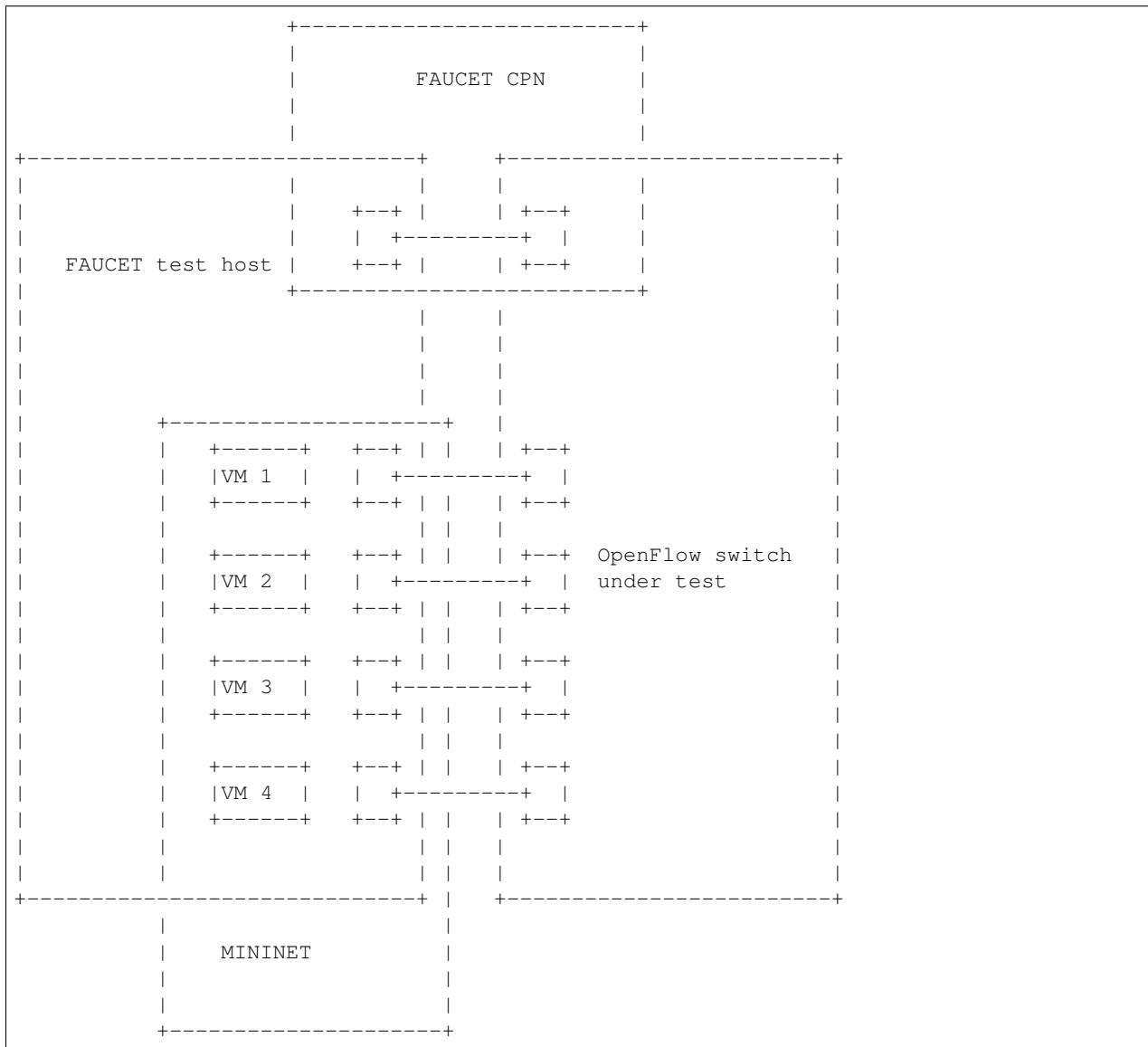
```

sudo modprobe openvswitch
sudo docker run --sysctl net.ipv6.conf.all.disable_ipv6=0 --privileged -ti faucet/
↪ tests

```

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

2.3.3 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).

NOTE: it is very important to disable any process that cause any traffic on the dataplane test interfaces, as this may interfere with the tests. For example, the test interfaces should have all IPv4/IPv6 dynamic address assignment disabled. To achieve this, on Ubuntu for example, you can set the interfaces to “unmanaged” in Network Manager, and make sure processes like *Avahi* ignores those interfaces.

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/faucet
$EDITOR /etc/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 --pull -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/faucet:/etc/faucet \
```

(continues on next page)

(continued from previous page)

```
-v /tmp:/tmp \  
-ti faucet/tests
```

Running a single test including pytype, linting, and documentation

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

Running only the integration tests

Sometimes you will want to skip the pytype, linting and documentation tests in order to complete a faucet test suite run against hardware quicker.

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

Running only a single integration test

Sometimes you will want to skip the pytype, linting and documentation tests and simply run a single integration test. Optionally `-k` will also save the results.

```
sudo docker run --privileged --net=host \  
-e FAUCET_TESTS="-i -n -k FaucetUntaggedLLDPTest" \  
-v /etc/faucet:/etc/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 config 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/config-fuzzer -f Dockerfile.fuzz-config .

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

AFL then will run indefinitely. You can find the output in /var/log/afl/. You will then need to run the output configs with faucet to see the error produced.

2.4.2 Fuzzing faucet packet handling with docker

Build and run the afl-fuzz tests:

```
docker build -t faucet/packet-fuzzer -f Dockerfile.fuzz-packet .

docker run -d \
    -u $(id -u $USER) \
    --name packet-fuzzer \
    -v /var/log/afl:/var/log/afl/ \
    -v /var/log/faucet:/var/log/faucet/ \
    -p 6653:6653 \
    -p 9302:9302 \
    faucet/packet-fuzzer
```

AFL will then fuzz the packet handling indefinitely. The afl output can be found in /var/log/afl/. To check the error produced by an afl crash file use `display_packet_crash`:

```
python3 tests/fuzzer/display_packet_crash.py /var/log/afl/crashes/X
```

Where X is the name of the crash file. The output can then be found in the faucet logs (/var/log/faucet/).

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`

Contains the state for an ACL, including the configuration.

ACL Config

ACLs are configured under the 'acls' configuration block. The acls block contains a dictionary of individual acls each keyed by its name.

Each acl contains a list of rules, a packet will have the first matching rule applied to it.

Each rule is a dictionary containing the single key 'rule' with the value the matches and actions for the rule.

The matches are key/values based on the ryu RESTful API. The key 'actions' contains a dictionary with keys/values as follows:

- allow (int): if 1 allow the packet to continue through the Faucet pipeline, if 0 drop the packet.
- force_port_vlan (int): if 1, do not verify the VLAN/port association for this packet and override any VLAN ACL on the forced VLAN.
- meter (str): meter to apply to the packet
- output (dict): used to output a packet directly. details below.
- cookie (int): set flow cookie to this value on this flow

The output action contains a dictionary with the following elements:

- port (int or string): the port to output the packet to
- ports (list): a list of the ports (int or string) to output the packet to
- set_fields (list): a list of fields to set with values
- dl_dst (str): old style request to set eth_dst to a value (set_fields recommended)
- pop_vlans: (int): pop the packet vlan before outputting
- vlan_vid: (int): push the vlan vid on the packet when outputting
- vlan_vids: (list): push the list of vlans on the packet when outputting, with option eth_type
- swap_vid (int): rewrite the vlan vid of the packet when outputting
- failover (dict): Output with a failover port (experimental)

```
actions_types = {'allow': <class 'int'>, 'force_port_vlan': <class 'int'>, 'meter':
defaults = {'exact_match': False, 'rules': None}
defaults_types = {'exact_match': <class 'bool'>, 'rules': <class 'list'>}
exact_match = None
output_actions_types = {'dl_dst': <class 'str'>, 'failover': <class 'dict'>, 'pop_vl':
rule_types = {'actions': <class 'dict'>, 'cookie': <class 'int'>, 'description': <c
rules = None
to_conf ()
    Return configuration as a dict.
```

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, debug_level, check_output_file)
```

Return True and successful config dict, if all config can be parsed.

```
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 ()
        Check config at instantiation time for errors, typically via assert.

    conf_hash (dyn=False, subconf=True, ignore_keys=None)
        Return hash of keys configurably filtering attributes.

    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.conf.test_config_condition (cond, msg)
```

faucet.config_parser module

Implement configuration file parsing.

```
faucet.config_parser.dp_parser (config_file, logname)
    Parse a config file into DP configuration objects with hashes of config include/files.

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.

class faucet.config_parser_util.UniqueKeyLoader (*stream*)

Bases: `yaml.loader.Loader`

construct_mapping (*node, deep=False*)

Check for duplicate YAML keys.

faucet.config_parser_util.config_changed (*top_config_file, new_top_config_file, config_hashes*)

Return True if configuration has changed.

Parameters

- **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 True if the file, or any file it includes, has changed.

Return type bool

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*)

Handles including additional config files

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`

Stores state related to a datapath controlled by Faucet, including configuration.

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.

advertise_interval = None

all_valve_tables()
Return list of all Valve tables.

arp_neighbor_timeout = None

bgp_vlans()
Return list of VLANs with BGP enabled.

check_config()
Check config at instantiation time for errors, typically via assert.

combinatorial_port_flood = None

configured = False

cookie = None

defaults = {'advertise_interval': 30, 'arp_neighbor_timeout': 250, 'combinatorial_po

defaults_types = {'advertise_interval': <class 'int'>, 'arp_neighbor_timeout': <clas

dp_acls = None

dp_id = None

drop_broadcast_source_address = None

drop_spoofed_faucet_mac = None

dyn_last_coldstart_time = None

faucet_dp_mac = None

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

get_config_changes(logger, new_dp)
Detect any config changes.

Parameters

- **logger** ([ValveLogger](#)) – logger instance
- **new_dp** ([DP](#)) – new dataplane configuration.

Returns

changes tuple containing:

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.

Return type (tuple)

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
lACP_timeout = None
learn_ban_timeout = None
learn_jitter = None
lldp_beacon = {}
lldp_beacon_defaults_types = {'max_per_interval': <class 'int'>, 'send_interval': <c
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 = {}
metrics_rate_limit_sec = None
name = None
output_only_ports = 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
reset_refs (vlans=None)
resolve_stack_topology (dps)
    Resolve inter-DP config for stacking.
routers = None
running = False
set_defaults ()
    Set default values and run any basic sanity checks.
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_defaults_types = {'priority': <class 'int'>}  
  
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.  
  
vlangs = 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.EventFaucetLLDPAdvertise  
    Bases: ryu.controller.event.EventBase  
  
    Event used to trigger periodic LLDP beacons.  
  
class faucet.faucet.EventFaucetMetricUpdate  
    Bases: ryu.controller.event.EventBase  
  
    Event used to trigger update of metrics.  
  
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: faucet.valve_ryuapp.RyuAppBase

    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.

    bgp = None

    desc_stats_reply_handler(ryu_event)
        Handle OFPDescStatsReply from datapath.

        Parameters ryu_event (ryu.controller.ofp_event.EventOFPDescStatsReply) – trigger.

    error_handler(ryu_event)
        Handle an OFPError from a datapath.

        Parameters 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.

        Parameters ryu_event (ryu.controller.ofp_event.EventOFPStateChange) – trigger.

    flowremoved_handler(ryu_event)
        Handle a flow removed event.

        Parameters 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.

    metrics = None

    notifier = None

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

        Parameters ryu_event (ryu.controller.event.EventReplyBase) – packet in message.

    port_status_handler(ryu_event)
        Handle a port status change event.

        Parameters ryu_event (ryu.controller.ofp_event.EventOFPPortStatus) – trigger.

    reload_config(ryu_event)
        Handle a request to reload configuration.

```

```
start ()  
    Start controller.  
  
valves_manager = None
```

faucet.faucet_bgp module

BGP implementation for FAUCET.

```
class faucet.faucet_bgp.FaucetBgp (logger, metrics, send_flow_msgs)  
    Bases: object  
  
    Wrapper for Ryu BGP speaker.  
  
    reset (valves)  
        Set up a BGP speaker for every VLAN that requires it.  
  
    shutdown_bgp_speakers ()  
        Shutdown any active BGP speakers.  
  
    update_metrics (_now)  
        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_experimental_event module

Experimental FAUCET event notification.

```
class faucet.faucet_experimental_event.FaucetExperimentalEventNotifier (socket_path,
                                                                    met-
                                                                    rics,
                                                                    log-
                                                                    ger)
```

Bases: object

Event notification, via Unix domain socket.

```
check_path (socket_path)
    Check that socket_path is valid.
```

```
notify (dp_id, dp_name, event_dict)
    Notify of an event.
```

```
start ()
    Start socket server.
```

```
class faucet.faucet_experimental_event.NonBlockLock
    Bases: object
```

Non blocking lock that can be used as a context manager.

```
acquire_nonblock ()
    Attempt to acquire a lock.
```

```
release ()
    Release lock when done.
```

faucet.faucet_metrics module

Implement Prometheus statistics.

```
class faucet.faucet_metrics.FaucetMetrics (reg=None)
    Bases: faucet.prom_client.PromClient
```

Container class for objects that can be exported to Prometheus.

```
reset_dpids (dp_labels)
    Set all DPID-only counter/gauges to 0.
```

faucet.faucet_pipeline module

Standard FAUCET pipeline.

faucet.fctl module

Report state based on FAUCET/Gauge/Prometheus variables.

```
faucet.fctl.decode_value (metric_name, value)
    Convert values to human readable format based on metric name
```

```
faucet.fctl.main ()
```

`faucet.fctl.parse_args(sys_args)`
 Parse and return CLI args.

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

`faucet.fctl.scrape_prometheus(endpoints, retries=3, err_output_file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>)`
 Scrape a list of Prometheus/FAUCET/Gauge endpoints and aggregate results.

faucet.gauge module

RyuApp shim between Ryu and Gauge.

class `faucet.gauge.Gauge(*args, **kwargs)`
 Bases: `faucet.valve_ryuapp.RyuAppBase`

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,

exc_logname = 'gauge.exception'

logname = 'gauge'

prom_client = None

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

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

Parameters *ryu_event* (*ryu.controller.event.EventReplyBase*) – port status change event.

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`

Example

```
> 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
```

(continues on next page)

(continued from previous page)

```

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*)

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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

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

Bases: `faucet.gauge_pollers.GaugePortStatePoller`, `faucet.gauge_influx.InfluxShipper`

Example

```

> 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

```

(continues on next page)

(continued from previous page)

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*)

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.**Parameters**

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

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.

Example

```
> 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
```

(continues on next page)

(continued from previous page)

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*)

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.**Parameters**

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

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_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 OF-FlowStatsReply message (in yaml format) that matches all flows.**no_response** ()

Called when a polling cycle passes without receiving a response.

send_req ()

Send a stats request to a datapath.

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

    Abstraction for a poller for statistics.

    is_active ()
        Return True if the poller is controlling the request loop for its stat

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

    report_dp_status (dp_status)
        Report DP status.

    running ()
        Return True if the poller is running.

    send_req ()
        Send a stats request to a datapath.

    start (ryudp, active)
        Start the poller.

    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.

        Parameters

        • rcv_time – the time the response was received

        • dp_id – DP ID

        • msg – the stats reply message

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

    Abstraction for port state poller.

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

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

    send_req ()
        Send a stats request to a datapath.

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.

```
is_active()
    Return True if the poller is controlling the request loop for its stat

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

send_req()
    Send a stats request to a datapath.

start(ryudp, active)
    Start the poller.

stop()
    Stop the poller.
```

faucet.gauge_prom module

Prometheus for Gauge.

```
class faucet.gauge_prom.GaugeFlowTablePrometheusPoller(conf, logname, prom_client)
    Bases: faucet.gauge_pollers.GaugeFlowTablePoller
```

Export flow table entries to Prometheus.

```
table_tags = {}
```

```
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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

```
class faucet.gauge_prom.GaugePortStatePrometheusPoller(conf, logname, prom_client)
    Bases: faucet.gauge_pollers.GaugePortStatePoller
```

Export port state changes to Prometheus.

```
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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID

- **msg** – the stats reply message

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*)

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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

class faucet.gauge_prom.**GaugePrometheusClient** (*reg=None*)
Bases: *faucet.prom_client.PromClient*

Wrapper for Prometheus client that is shared between all pollers.

metrics = {}

reregister_flow_vars (*table_name, table_tags*)

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

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

entry = None

entry_msg = None

meter_id = None

faucet.port module

Port configuration.

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

Stores state for ports, including the configuration.

acl_in = None

acls_in = None

check_config()

Check config at instantiation time for errors, typically via assert.

defaults = {'acl_in': None, 'acls_in': None, 'description': None, 'enabled': True,

defaults_types = {'acl_in': (<class 'str'>, <class 'int'>), 'acls_in': <class 'list'>

description = None

dp_id = None

dyn_lacp_up = None

dyn_lacp_updated_time = None

dyn_last_ban_time = None

dyn_last_lacp_pkt = None

dyn_last_lldp_beacon_time = None

dyn_learn_ban_count = 0

dyn_phys_up = False

enabled = None

finalize()

Configuration parsing marked complete.

hairpin = None

hosts (*vlan=None*)

Return all host cache entries this port has learned (on all or specified VLANs).

hosts_count (*vlan=None*)

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

lldp_beacon = {}

lldp_beacon_defaults_types = {'enable': <class 'bool'>, 'org_tlvs': <class 'list'>,

lldp_beacon_enabled()

Return True if LLDP beacon enabled on this port.

lldp_org_tlv_defaults_types = {'info': (<class 'str'>, <class 'bytearray'>), 'oui':

loop_protect = None

max_hosts = None

mirror = None

mirror_actions()

Return OF actions to mirror this port.

name = None

native_vlan = None

number = None

op_status_reconf = None

output_only = None

override_output_port = None

permanent_learn = None

```
receive_lldp = None

running()
    Return True if port enabled and up.

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

stack = {}

stack_defaults_types = {'dp': <class 'str'>, 'port': (<class 'str'>, <class 'int'>)}

tagged_vlans = []

to_conf()
    Return configuration as a dict.

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(reg=None)
    Bases: object

    Prometheus client.

    REQUIRED_LABELS = ['dp_id', 'dp_name']

    server = None

    start(prom_port, prom_addr, use_test_thread=False)
        Start webserver.

    thread = None

faucet.prom_client.make_wsgi_app(registry)
    Create a WSGI app which serves the metrics from a registry.
```

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()
        Check config at instantiation time for errors, typically via assert.

    defaults = {'vlans': None}

    defaults_types = {'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

    Serialize table features messages from JSON.

    load_tables (active_table_ids)

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.AlliedTelesis (dp, logname, metrics, notifier)
    Bases: faucet.valve.OVSValve

    Valve implementation for AT.

    DEC_TTL = False

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

    Valve implementation that uses OpenFlow send table features messages.

    DEC_TTL = False

    PIPELINE_CONF = 'aruba_pipeline.json'

class faucet.valve.OVSValve (dp, logname, metrics, notifier)
    Bases: faucet.valve.Valve

    Valve implementation for OVS.

    USE_BARRIERS = False

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

    Valve implementation that uses OpenFlow send table features messages.

    PIPELINE_CONF = 'tfm_pipeline.json'

    SKIP_VALIDATION_TABLES = ()

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

        Parameters msg (OFPSwitchFeatures) – msg sent from switch.

        Vendor specific configuration should be implemented here.

class faucet.valve.Valve (dp, logname, metrics, notifier)
    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**

USE_BARRIERS = **True**

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

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

base_prom_labels = **None**

close_logs ()
Explicitly close any active loggers.

datapath_connect (*now, discovered_ports*)
Handle Ryu datapath connection event and provision pipeline.

Parameters

- **now** (*float*) – current epoch time.
- **discovered_ports** (*list*) – datapath OFPorts.

Returns OpenFlow messages to send to datapath.

Return type list

datapath_disconnect ()
Handle Ryu datapath disconnection event.

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

dp_init ()
Initialize datapath state at connection/re/config time.

flood_manager = **None**

flow_timeout (*now, table_id, match*)
Call flow timeout message handler:

Parameters

- **now** (*float*) – current epoch time.
- **table_id** (*int*) – ID of table where flow was installed.
- **match** (*dict*) – match conditions for expired flow.

Returns OpenFlow messages, if any.

Return type list

get_config_dict ()
Return datapath config as a dict for experimental API.

host_manager = **None**

lACP_down (*port*)
Return OpenFlow messages when LACP is down on a port.

lACP_handler (*now, pkt_meta*)

Handle a LACP packet.

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

Parameters

- **now** (*float*) – current epoch time.
- **pkt_meta** (*PacketMeta*) – packet for control plane.

Returns OpenFlow messages, if any.

Return type list

lACP_up (*port*)

Return OpenFlow messages when LACP is up on a port.

lldp_handler (*pkt_meta*)

Handle an LLDP packet.

Parameters **pkt_meta** (*PacketMeta*) – packet for control plane.

logger = None

ofchannel_log (*ofmsgs*)

Log OpenFlow messages in text format to debugging log.

ofchannel_logger = None

ofdescstats_handler (*body*)

Handle OF DP description.

oferror (*msg*)

Correlate OFError message with flow we sent, if any.

Parameters **msg** (*ryu.controller.ofp_event.EventOFPMsgBase*) – message from datapath.

parse_pkt_meta (*msg*)

Parse OF packet-in message to PacketMeta.

parse_rcv_packet (*in_port, vlan_vid, eth_type, data, orig_len, pkt, eth_pkt*)

Parse a received packet into a PacketMeta instance.

Parameters

- **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.
- **eth_pkt** (*ryu.lib.packet.ethernet*) – parsed Ethernet header.

Returns PacketMeta instance.

port_add (*port_num*)

Handle addition of a single port.

Parameters **port_num** (*list*) – list of port numbers.

Returns OpenFlow messages, if any.

Return type list

port_delete (*port_num*)

Return flow messages that delete port from pipeline.

port_no_valid (*port_no*)

Return True if supplied port number valid on this datapath.

port_status_handler (*port_no, reason, state*)

Return OpenFlow messages responding to port operational status change.

ports_add (*port_nums, cold_start=False, log_msg='up'*)

Handle the addition of ports.

Parameters

- **port_num** (*list*) – list of port numbers.
- **cold_start** (*bool*) – True if configuring datapath from scratch.

Returns OpenFlow messages, if any.

Return type list

ports_delete (*port_nums, log_msg='down'*)

Handle the deletion of ports.

Parameters **port_nums** (*list*) – list of port numbers.

Returns OpenFlow messages, if any.

Return type list

prepare_send_flows (*flow_msgs*)

Prepare to send flows to datapath.

Parameters **flow_msgs** (*list*) – OpenFlow messages to send.

rate_limit_packet_ins (*now*)

Return True if too many packet ins this second.

rcv_packet (*now, 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.

Parameters

- **other_valves** (*list*) – all Valves other than this one.
- **pkt_meta** (*PacketMeta*) – packet for control plane.

Returns OpenFlow messages, if any.

Return type list

recent_ofmsgs = **deque** ([], **maxlen**=32)

reload_config (*now, 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...

Parameters

- **now** (*float*) – current epoch time.
- **new_dp** (*DP*) – new dataplane configuration.

Returns OpenFlow messages.

Return type ofmsgs (list)

resolve_gateways (*now*)

Call route managers to re/resolve gateways.

Returns OpenFlow messages, if any.

Return type list

send_flows (*ryu_dp, flow_msgs*)

Send flows to datapath.

Parameters

- **ryu_dp** (*ryu.controller.controller.Datapath*) – datapath.
- **flow_msgs** (*list*) – OpenFlow messages to send.

send_lldp Beacons (*now*)

Called periodically to send LLDP beacon packets.

state_expire (*now*)

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

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

Returns OpenFlow messages, if any.

Return type list

switch_features (*_msg*)

Send configuration flows necessary for the switch implementation.

Parameters **msg** (*OFPSwitchFeatures*) – msg sent from switch.

Vendor specific configuration should be implemented here.

update_config_metrics ()

Update gauge/metrics for configuration.

update_metrics (*now, updated_port=None, rate_limited=False*)

Update Gauge/metrics.

class faucet.valve.ValveLogger (*logger, dp_id*)

Bases: object

Logger for a Valve that adds DP ID.

debug (*log_msg*)

Log debug level message.

error (*log_msg*)

Log error level message.

info (*log_msg*)
Log info level message.

warning (*log_msg*)
Log warning level message.

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

Parameters `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*, *meters*, *acl_allow_inst*, *acl_force_port_vlan_inst*,
port_num=None, *vlan_vid=None*)

`faucet.valve_acl.build_acl_ofmsgs` (*acls*, *acl_table*, *acl_allow_inst*, *acl_force_port_vlan_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*, *eth_src_table*, *flood_priority*, *bypass_priority*, *use_group_table*, *groups*, *combinatorial_port_flood*)

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.

Parameters

- **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,          eth_src_table,
                                                flood_priority,        bypass_priority,
                                                use_group_table,    groups,    combina-
                                                torial_port_flood, stack, stack_ports,
                                                dp_shortest_path_to_root,    short-
                                                est_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.

Parameters

- **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)
```

Expire hosts from VLAN cache.

```
flow_timeout (now, table_id, match)
```

Handle a flow timed out message from dataplane.

```
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

Manage host learning on VLANs.

```
CACHE_UPDATE_GUARD_TIME = 2
```

```
ban_rules (pkt_meta)
```

Limit learning to a maximum configured on this port/VLAN.

Parameters **pkt_meta** – PacketMeta instance.

Returns OpenFlow messages, if any.

Return type list

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 (*_now*, *_table_id*, *_match*)

Handle a flow timed out message from dataplane.

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 (*now*, *port*, *vlan*, *eth_src*, *delete_existing=True*,
last_dp_coldstart_time=None)

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.

Parameters *actions* (*list*) – list of OpenFlow actions.

Returns instruction of actions.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPInstruction

faucet.valve_of.apply_meter (*meter_id*)

Return instruction to apply a meter.

faucet.valve_of.barrier ()

Return OpenFlow barrier request.

Returns barrier request.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPBarrierRequest

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*, *icmpv6_type=None*,
nw_proto=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 decrement IP TTL.

Return type `ryu.ofproto.ofproto_v1_3_parser.OFPActionDecNwTtl`

`faucet.valve_of.dedupe_ofmsgs` (*input_ofmsgs*)

Return deduplicated ofmsg list.

`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.

Parameters `vid` (*int*) – VLAN VID with VID_PRESENT.

Returns VLAN VID.

Return type `int`

`faucet.valve_of.faucet_async` (*datapath=None, notify_flow_removed=False, packet_in=True*)

Return async message config for FAUCET/Gauge

`faucet.valve_of.faucet_config` (*datapath=None*)

Return switch config for FAUCET.

`faucet.valve_of.flood_tagged_port_outputs` (*ports, in_port=None, 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=None, 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.goto_table` (*table*)

Return instruction to goto table.

Parameters `table` (`ValveTable`) – table to goto.

Returns goto instruction.

Return type `ryu.ofproto.ofproto_v1_3_parser.OFPInstruction`

`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.

Parameters `port_num` (*int*) – switch port.

Returns True if FAUCET should ignore this port.

Return type bool

`faucet.valve_of.is_delete(ofmsg)`

`faucet.valve_of.is_flowdel(ofmsg)`

Return True if flow message is a FlowMod and a delete.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a FlowMod delete/strict.

Return type bool

`faucet.valve_of.is_flowmod(ofmsg)`

Return True if flow message is a FlowMod.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a FlowMod

Return type bool

`faucet.valve_of.is_groupadd(ofmsg)`

Return True if OF message is a GroupMod and command is add.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a GroupMod add

Return type bool

`faucet.valve_of.is_groupdel(ofmsg)`

Return True if OF message is a GroupMod and command is delete.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a GroupMod delete

Return type bool

`faucet.valve_of.is_groupmod(ofmsg)`

Return True if OF message is a GroupMod.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a GroupMod

Return type bool

`faucet.valve_of.is_meteradd(ofmsg)`

Return True if OF message is a MeterMod and command is add.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a MeterMod add

Return type bool

`faucet.valve_of.is_meterdel(ofmsg)`

Return True if OF message is a MeterMod and command is delete.

Parameters *ofmsg* – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a MeterMod delete

Return type bool

`faucet.valve_of.is_metermod(ofmsg)`

Return True if OF message is a MeterMod.

Parameters `ofmsg` – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a MeterMod

Return type bool

`faucet.valve_of.is_table_features_req(ofmsg)`

Return True if flow message is a TFM req.

Parameters `ofmsg` – ryu.ofproto.ofproto_v1_3_parser message.

Returns True if is a TFM req.

Return type bool

`faucet.valve_of.match(match_fields)`

Return OpenFlow matches from dict.

Parameters `match_fields` (*dict*) – match fields and values.

Returns matches.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPMatch

`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.

Parameters `max_len` (*int*) – max number of bytes from packet to output.

Returns packet in action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput

`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.

Parameters

- `port_num` (*int*) – port to output to.
- `max_len` (*int*) – maximum length of packet to output (default no maximum).

Returns output to port action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput

`faucet.valve_of.packetout(port_num, data)`

Return OpenFlow action to packet out to dataplane from controller.

Parameters

- `port_num` (*int*) – port to output to.

- **data** (*str*) – raw packet to output.

Returns packet out action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput

`faucet.valve_of.pop_vlan()`

Return OpenFlow action to pop outermost Ethernet 802.1Q VLAN header.

Returns Pop VLAN.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionPopVlan

`faucet.valve_of.port_status_from_state(state)`

Return True if OFPPS_LINK_DOWN is not set.

`faucet.valve_of.push_vlan_act(vlan_vid, eth_type=33024)`

Return OpenFlow action list to push Ethernet 802.1Q header with VLAN VID.

Parameters **vid** (*int*) – VLAN VID

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

Return type list

`faucet.valve_of.set_eth_dst(eth_dst)`

Return action to set destination Ethernet MAC address.

Parameters **eth_src** (*str*) – destination Ethernet MAC address.

Returns set field action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

`faucet.valve_of.set_eth_src(eth_src)`

Return action to set source Ethernet MAC address.

Parameters **eth_src** (*str*) – source Ethernet MAC address.

Returns set field action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

`faucet.valve_of.set_field(**kws)`

Return action to set any field.

Parameters **kws** (*dict*) – exactly one field to set

Returns set field action.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

`faucet.valve_of.set_vlan_vid(vlan_vid)`

Set VLAN VID with VID_PRESENT flag set.

Parameters **vid** (*int*) – VLAN VID

Returns set VID with VID_PRESENT.

Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

`faucet.valve_of.table_features(body)`

`faucet.valve_of.valve_flowreorder(input_ofmsgs, use_barriers=True)`

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.

Parameters `vid(int)` – VLAN VID

Returns VLAN VID with VID_PRESENT.

Return type `int`

`faucet.valve_of_old` module

Deprecated OF matches.

`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 at 0x7effef06d950>, <class 'r

MIN_ETH_TYPE_PKT_SIZE = {2048: 38, 2054: 46, 34525: 58}

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(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.

Parameters

- **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 serialized ARP reply packet.

Return type `ryu.lib.packet.arp`

`faucet.valve_packet.arp_request(vid, eth_src, src_ip, dst_ip)`

Return an ARP request packet.

Parameters

- **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 serialized ARP request packet.

Return type `ryu.lib.packet.arp`

`faucet.valve_packet.build_pkt_header(vid, eth_src, eth_dst, dl_type)`

Return an Ethernet packet header.

Parameters

- **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 Ethernet packet with header.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.echo_reply(vid, eth_src, eth_dst, src_ip, dst_ip, data)`

Return an ICMP echo reply packet.

Parameters

- **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 serialized ICMP echo reply packet.

Return type `ryu.lib.packet.icmp`

`faucet.valve_packet.faucet_lldp_tlvs(dp)`

Return LLDP TLVs for a datapath.

`faucet.valve_packet.faucet_oui(mac)`

Return first 3 bytes of MAC address (given as str).

`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.

Parameters

- **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 Serialized IPv6 ICMP echo reply packet.

Return type `ryu.lib.packet.ethernet`

`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.

Parameters **dst_ip** (*ipaddress.IPv6Address*) – IPv6 address.

Returns Ethernet multicast address.

Return type `str`

`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.

Parameters **ucast** (*ipaddress.IPv6Address*) – IPv6 unicast address.

Returns IPv6 solicited node multicast address.

Return type `ipaddress.IPv6Address`

`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.

Parameters

- **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 Ethernet packet with header.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.lldp_beacon(eth_src, chassis_id, port_id, ttl, org_tlvs=None, system_name=None, port_descr=None)`

Return an LLDP frame suitable for a host/access port.

Parameters

- **eth_src**(*str*) – source Ethernet MAC address.
- **chassis_id**(*str*) – Chassis ID.
- **port_id**(*int*) – port ID,
- **TTL**(*int*) – TTL for payload.
- **org_tlvs**(*list*) – list of tuples of (OUI, subtype, info).

Returns Ethernet packet with header.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.mac_addr_is_unicast(mac_addr)`

Returns True if `mac_addr` is a unicast Ethernet address.

Parameters **mac_addr**(*str*) – MAC address.

Returns True if a unicast Ethernet address.

Return type `bool`

`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 advertisement packet.

Parameters

- **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 Serialized IPv6 neighbor discovery packet.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.nd_request(vid, eth_src, src_ip, dst_ip)`

Return IPv6 neighbor discovery request packet.

Parameters

- **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 Serialized IPv6 neighbor discovery packet.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.parse_eth_pkt(pkt)`

Return parsed Ethernet packet.

Parameters **pkt** (*ryu.lib.packet.packet*) – packet received from dataplane.

Returns Ethernet packet.

Return type `ryu.lib.packet.ethernet`

`faucet.valve_packet.parse_lacp_pkt(pkt)`

Return parsed LACP packet.

Parameters **pkt** (*ryu.lib.packet.packet*) – packet received from dataplane.

Returns LACP packet.

Return type `ryu.lib.packet.lacp`

`faucet.valve_packet.parse_lldp(pkt)`

Return parsed LLDP packet.

Parameters **pkt** (*ryu.lib.packet.packet*) – packet received from dataplane.

Returns LLDP packet.

Return type `ryu.lib.packet.lldp`

`faucet.valve_packet.parse_packet_in_pkt(data, max_len)`

Parse a packet received via packet in from the dataplane.

Parameters

- **data** (*bytearray*) – packet data from dataplane.
- **max_len** (*int*) – max number of packet data bytes to parse.

Returns raw packet `ryu.lib.packet.ethernet`: parsed Ethernet packet. `int`: Ethernet type of packet (inside VLAN) `int`: VLAN VID (or None if no VLAN)

Return type `ryu.lib.packet.packet`

`faucet.valve_packet.parse_vlan_pkt(pkt)`

Return parsed VLAN header.

Parameters **pkt** (*ryu.lib.packet.packet*) – packet received from dataplane.

Returns VLAN header.

Return type `ryu.lib.packet.vlan`

`faucet.valve_packet.router_advert(vid, eth_src, eth_dst, src_ip, dst_ip, vips, pi_flags=6)`

Return IPv6 ICMP echo reply packet.

Parameters

- **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 Serialized IPv6 ICMP RA packet.

Return type `ryu.lib.packet.ethernet`

faucet.valve_route module

Valve IPv4/IPv6 routing implementation.

class `faucet.valve_route.NextHop` (*eth_src, port, now*)

Bases: `object`

Describes a directly connected (at layer 2) nexthop.

age (*now*)

Return age of this nexthop.

dead (*max_fib_retries*)

Return True if this nexthop is considered dead.

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

advertise (*_vlan*)

control_plane_handler (*now, pkt_meta*)

resolve_gw_on_port (*vlan, port, faucet_vip, ip_gw*)

resolve_gw_on_vlan (*vlan, faucet_vip, ip_gw*)

```
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 (now, pkt_meta)
```

```
resolve_gw_on_port (vlan, port, faucet_vip, ip_gw)
```

```
resolve_gw_on_vlan (vlan, faucet_vip, ip_gw)
```

```
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 = ()
```

```
ETH_TYPE = None
```

```
ICMP_TYPE = None
```

```
IPV = 0
```

```
MAX_LEN = 128
```

```
active = False
```

```
add_faucet_vip (vlan, faucet_vip)
```

```
add_host_fib_route_from_pkt (now, pkt_meta)
```

Add a host FIB route given packet from host.

Parameters

- **now** (*float*) – seconds since epoch.
- **pkt_meta** (*PacketMeta*) – received packet.

Returns OpenFlow messages.

Return type list

add_route (*vlan*, *ip_gw*, *ip_dst*)

Add a route to the RIB.

Parameters

- **vlan** (*vlan*) – VLAN containing this RIB.
- **ip_gw** (*ipaddress.ip_address*) – IP address of nexthop.
- **ip_dst** (*ipaddress.ip_network*) – destination IP network.

Returns OpenFlow messages.

Return type list

advertise (*vlan*)

control_plane_handler (*now*, *pkt_meta*)

del_route (*vlan*, *ip_dst*)

Delete a route from the RIB.

Only one route with this exact destination is supported.

Parameters

- **vlan** (*vlan*) – VLAN containing this RIB.
- **ip_dst** (*ipaddress.ip_network*) – destination IP network.

Returns OpenFlow messages.

Return type list

nexthop_dead (*nexthop_cache_entry*)

resolve_gateways (*vlan*, *now*)

Re/resolve all gateways.

Parameters

- **vlan** (*vlan*) – VLAN containing this RIB/FIB.
- **now** (*float*) – seconds since epoch.

Returns OpenFlow messages.

Return type list

resolve_gw_on_port (*vlan*, *port*, *faucet_vip*, *ip_gw*)

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

faucet.valve_ryuapp module

RyuApp base class for FAUCET/Gauge.

class faucet.valve_ryuapp.**EventReconfigure**

Bases: ryu.controller.event.EventBase

Event sent to controller to cause config reload.

class faucet.valve_ryuapp.**RyuAppBase** (**args*, ***kwargs*)

Bases: ryu.base.app_manager.RyuApp

RyuApp base class for FAUCET/Gauge.

```

OFF_VERSIONS = [4]

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

    Parameters ryu_event (ryu.controller.dpset.EventDP) – trigger.

exc_logname = ''

get_setting(setting, path_eval=False)
    Return config setting prefaced with logname.

logname = ''

reconnect_handler(ryu_event)
    Handle reconnection of a datapath.

    Parameters ryu_event (ryu.controller.dpset.EventDPReconnected) – trigger.

reload_config(_ryu_event)
    Handle reloading configuration.

signal_handler(sigid, _)
    Handle signals.

    Parameters sigid (int) – signal received.

start()
    Start controller.

```

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, cookie=None*)

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, icmpv6_type=None, nw_proto=None, nw_dst=None*)

Compose an OpenFlow match rule.

faucet.valve_util module

Utility functions for FAUCET.

faucet.valve_util.**btoa** (*b_str*)

Return byte array/string as string.

faucet.valve_util.**close_logger** (*logger*)

Close all handlers on logger object.

faucet.valve_util.**dpid_log** (*dpid*)

Log a DP ID as hex/decimal.

faucet.valve_util.**get_logger** (*logname, logfile, loglevel, propagate*)

Create and return a logger object.

faucet.valve_util.**get_setting** (*name, path_eval=False*)

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.valves_manager module

Manage a collection of Valves.

class faucet.valves_manager.**ConfigWatcher**

Bases: object

Watch config for file or content changes.

config_file = None

config_file_stats = None

config_hashes = None

content_changed (*new_config_file*)

Return True if config file content actually changed.

files_changed ()

Return True if any config files changed.

update (*new_config_file*, *new_config_hashes=None*)

Update state with new config file/hashes.

class faucet.valves_manager.ValvesManager (*logname, logger, metrics, notifier, bgp,*
send_flows_to_dp_by_id)

Bases: object

Manage a collection of Valves.

load_configs (*now, new_config_file, delete_dp=None*)

Load/apply new config to all Valves.

new_valve (*new_dp*)

parse_configs (*new_config_file*)

Return parsed configs for Valves, or None.

request_reload_configs (*now, new_config_file, delete_dp=None*)

Process a request to load config changes.

update_metrics (*now*)

Update metrics in all Valves.

valve_flow_services (*now, valve_service*)

Call a method on all Valves and send any resulting flows.

valve_packet_in (*now, valve, pkt_meta*)

Time a call to Valve packet in handler.

valves = {}

faucet.vlan module

VLAN configuration.

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

Bases: object

Association of a host with a port.

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

Bases: *faucet.conf.Conf*

Contains state for one VLAN, including its configuration.

acl_in = None

acls_in = None

add_cache_host (*eth_src, port, cache_time*)

add_route (*ip_dst*, *ip_gw*)

Add an IP route.

all_ip_gws (*ipv*)

Return list of all IP gateways for specified IP version.

bgp_as = None

bgp_connect_mode = None

bgp_ipvs ()

Return list of IP versions for BGP configured on this VLAN.

bgp_local_address = None

bgp_neighbor_addresses = []

bgp_neighbor_addresses_by_ipv (*ipv*)

Return list of BGP neighbor addresses with specified IP version on this VLAN.

bgp_neighbor_as = None

bgp_neighbour_addresses = []

bgp_neighbour_as = None

bgp_port = None

bgp_routerid = None

bgp_server_addresses = []

bgp_server_addresses_by_ipv (*ipv*)

Return list of BGP server addresses with specified IP version on this VLAN.

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_count_on_port (*port*)

Return count of all hosts learned on a port.

cached_hosts_on_port (*port*)

Return all hosts learned on a port.

check_config ()

Check config at instantiation time for errors, typically via assert.

clear_cache_hosts_on_port (*port*)

Clear all hosts learned on a port.

defaults = {'acl_in': None, 'acls_in': None, 'bgp_as': None, 'bgp_connect_mode': 'I

defaults_types = {'acl_in': (<class 'int'>, <class 'str'>), 'acls_in': <class 'list'>

del_route (*ip_dst*)

Delete an IP route.

dp_id = None

dyn_bgp_neighbor_addresses_by_ipv = None

dyn_bgp_server_addresses_by_ipv = None

dyn_faucet_vips_by_ipv = None

dyn_gws_by_ipv = None

dyn_host_cache = None
dyn_host_cache_by_port = None
dyn_last_time_hosts_expired = None
dyn_learn_ban_count = 0
dyn_neigh_cache_by_ipv = None
dyn_oldest_host_time = None
dyn_routes_by_ipv = None
expire_cache_host (*eth_src*)
 Expire a host from caches.
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, random_order, *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.
Parameters

- **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.
hosts_count ()
 Return number of hosts learned on this VLAN.
ip_dsts_for_ip_gw (*ip_gw*)
 Return list of IP destinations, for specified gateway.
ip_in_vip_subnet (*ipa*)
 Return *faucet_vip* if IP in same IP network as a 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

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

name = None

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

neigh_cache_count_by_ip (*ipv*)
Return number of hosts in neighbor cache for specified IP version on this VLAN.

output_port (*port*, *hairpin=False*)

pkt_out_port (*packet_builder*, *port*, **args*)

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_caches ()
Reset dynamic caches.

reset_ports (*ports*)

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

routes = None

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

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

tagged = None

tagged_flood_ports (*exclude_unicast*)

targeted_gw_resolution = None

to_conf ()
Return configuration as a dict.

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 OF-FlowStatsReply 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*)

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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

class faucet.watcher.**GaugePortStateLogger** (*conf, logname, prom_client*)

Bases: *faucet.gauge_pollers.GaugePortStatePoller*

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*)

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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID
- **msg** – the stats reply message

class faucet.watcher.**GaugePortStatsLogger** (*conf, logname, prom_client*)

Bases: *faucet.gauge_pollers.GaugePortStatsPoller*

Abstraction for port statistics logger.

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.

Parameters

- **rcv_time** – the time the response was received
- **dp_id** – DP ID

- **msg** – the stats reply message

`faucet.watcher.watcher_factory(conf)`

Return a Gauge object based on type.

Parameters `conf` (*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`

Stores the state and configuration to monitor a single stat.

Watcher Config

Watchers are configured in the watchers config block in the config for gauge.

The following elements can be configured for each watcher, at the level of `/watchers/<watcher name>/:`

- **type** (string): The type of watcher (IE what stat this watcher monitors). The types are ‘port_state’, ‘port_stats’ or ‘flow_table’.
- **dps** (list): A list of dps that should be monitored with this watcher.
- **db** (string): The db that will be used to store the data once it is retrieved.
- **interval** (int): if this watcher requires polling the switch, it will monitor at this interval.

The config for a db should be created in the gauge config file under the dbs config block.

The following elements can be configured for each db, at the level of `/dbs/<db name>/:`

- **type** (string): the type of db. The available types are ‘text’ and ‘influx’ for port_state, ‘text’, ‘influx’ and ‘prometheus’ for port_stats and ‘text’ and flow_table.

The following config elements then depend on the type. For text:

- **file** (string): the filename of the file to write output to.
- **compress** (bool): compress (with gzip) flow_table output while writing it

For influx:

- **influx_db** (str): The name of the influxdb database. Defaults to ‘faucet’.
- **influx_host** (str): The host where the influxdb is reachable. Defaults to ‘localhost’.
- **influx_port** (int): The port that the influxdb host will listen on. Defaults to 8086.
- **influx_user** (str): The username for accessing influxdb. Defaults to ‘’.
- **influx_pwd** (str): The password for accessing influxdb. Defaults to ‘’.
- **influx_timeout** (int): The timeout in seconds for connecting to influxdb. Defaults to 10.
- **influx_retries** (int): The number of times to retry connecting to influxdb after failure. Defaults to 3.

For Prometheus:

- **prometheus_port** (int): The port used to export prometheus data. Defaults to 9303.
- **prometheus_addr** (ip addr str): The address used to export prometheus data. Defaults to ‘127.0.0.1’.

```
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 = {'all_dps': False, 'compress': False, 'db': None, 'db_type': 'text', 'd
```

```
defaults_types = {'all_dps': <class 'bool'>, 'compress': <class 'bool'>, 'db': <cla
```

```
dp = None
```

```
prom_client = None
```

Module contents

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`

f

- [faucet](#), 145
- [faucet.acl](#), 98
- [faucet.check_faucet_config](#), 99
- [faucet.conf](#), 100
- [faucet.config_parser](#), 100
- [faucet.config_parser_util](#), 101
- [faucet.dp](#), 101
- [faucet.faucet](#), 104
- [faucet.faucet_bgp](#), 106
- [faucet.faucet_experimental_api](#), 106
- [faucet.faucet_experimental_event](#), 107
- [faucet.faucet_metrics](#), 107
- [faucet.faucet_pipeline](#), 107
- [faucet.fctl](#), 107
- [faucet.gauge](#), 108
- [faucet.gauge_influx](#), 108
- [faucet.gauge_pollers](#), 111
- [faucet.gauge_prom](#), 113
- [faucet.meter](#), 114
- [faucet.port](#), 114
- [faucet.prom_client](#), 116
- [faucet.router](#), 116
- [faucet.tfm_pipeline](#), 117
- [faucet.valve](#), 117
- [faucet.valve_acl](#), 122
- [faucet.valve_flood](#), 122
- [faucet.valve_host](#), 123
- [faucet.valve_of](#), 124
- [faucet.valve_of_old](#), 129
- [faucet.valve_packet](#), 129
- [faucet.valve_route](#), 134
- [faucet.valve_ryuapp](#), 136
- [faucet.valve_table](#), 137
- [faucet.valve_util](#), 138
- [faucet.valves_manager](#), 138
- [faucet.vlan](#), 139
- [faucet.watcher](#), 142
- [faucet.watcher_conf](#), 144

A

- ACL (class in faucet.acl), 98
- acl_in (faucet.port.Port attribute), 114
- acl_in (faucet.vlan.VLAN attribute), 139
- acls (faucet.dp.DP attribute), 101
- acls_in (faucet.port.Port attribute), 114
- acls_in (faucet.vlan.VLAN attribute), 139
- acquire_nonblock() (faucet.faucet_experimental_event.NonBlockLock method), 107
- actions_types (faucet.acl.ACL attribute), 99
- active (faucet.valve_route.ValveRouteManager attribute), 135
- add() (faucet.valve_table.ValveGroupEntry method), 137
- add_acl() (faucet.dp.DP method), 101
- add_cache_host() (faucet.vlan.VLAN method), 139
- add_db() (faucet.watcher_conf.WatcherConf method), 144
- add_dp() (faucet.watcher_conf.WatcherConf method), 145
- add_faucet_vip() (faucet.valve_route.ValveRouteManager method), 135
- add_host_fib_route_from_pkt() (faucet.valve_route.ValveRouteManager method), 135
- add_port() (faucet.dp.DP method), 101
- add_port_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
- add_route() (faucet.valve.Valve method), 118
- add_route() (faucet.valve_route.ValveRouteManager method), 135
- add_route() (faucet.vlan.VLAN method), 139
- add_router() (faucet.dp.DP method), 101
- add_vlan_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
- advertise() (faucet.valve.Valve method), 118
- advertise() (faucet.valve_route.ValveIPv4RouteManager method), 134
- advertise() (faucet.valve_route.ValveIPv6RouteManager method), 135
- advertise() (faucet.valve_route.ValveRouteManager method), 136
- advertise_interval (faucet.dp.DP attribute), 101
- age() (faucet.valve_route.NextHop method), 134
- all_dps (faucet.watcher_conf.WatcherConf attribute), 145
- all_ip_gws() (faucet.vlan.VLAN method), 140
- all_valve_tables() (faucet.dp.DP method), 101
- AlliedTelesis (class in faucet.valve), 117
- apply_actions() (in module faucet.valve_of), 124
- apply_meter() (in module faucet.valve_of), 124
- arp_neighbor_timeout (faucet.dp.DP attribute), 102
- arp_reply() (in module faucet.valve_packet), 129
- arp_request() (in module faucet.valve_packet), 129
- ArubaValve (class in faucet.valve), 117

B

- ban_rules() (faucet.valve_host.ValveHostManager method), 123
- barrier() (in module faucet.valve_of), 124
- base_prom_labels (faucet.valve.Valve attribute), 118
- bgp (faucet.faucet.Faucet attribute), 105
- bgp_as (faucet.vlan.VLAN attribute), 140
- bgp_connect_mode (faucet.vlan.VLAN attribute), 140
- bgp_ipvs() (faucet.vlan.VLAN method), 140
- bgp_local_address (faucet.vlan.VLAN attribute), 140
- bgp_neighbor_addresses (faucet.vlan.VLAN attribute), 140
- bgp_neighbor_addresses_by_ip() (faucet.vlan.VLAN method), 140
- bgp_neighbor_as (faucet.vlan.VLAN attribute), 140
- bgp_neighbour_addresses (faucet.vlan.VLAN attribute), 140
- bgp_neighbour_as (faucet.vlan.VLAN attribute), 140
- bgp_port (faucet.vlan.VLAN attribute), 140
- bgp_routerid (faucet.vlan.VLAN attribute), 140
- bgp_server_addresses (faucet.vlan.VLAN attribute), 140
- bgp_server_addresses_by_ip() (faucet.vlan.VLAN method), 140
- bgp_vlans() (faucet.dp.DP method), 102
- btos() (in module faucet.valve_util), 138

bucket() (in module faucet.valve_of), 124
build_acl_entry() (in module faucet.valve_acl), 122
build_acl_ofmsgs() (in module faucet.valve_acl), 122
build_flood_rules() (faucet.valve_flood.ValveFloodManager method), 122
build_flood_rules() (faucet.valve_flood.ValveFloodStackManager method), 123
build_match_dict() (in module faucet.valve_of), 124
build_output_actions() (in module faucet.valve_acl), 122
build_pkt_header() (in module faucet.valve_packet), 130

C

CACHE_UPDATE_GUARD_TIME

(faucet.valve_host.ValveHostManager attribute), 123
cached_host() (faucet.vlan.VLAN method), 140
cached_host_on_port() (faucet.vlan.VLAN method), 140
cached_hosts_count_on_port() (faucet.vlan.VLAN method), 140
cached_hosts_on_port() (faucet.vlan.VLAN method), 140
check_config() (faucet.conf.Conf method), 100
check_config() (faucet.dp.DP method), 102
check_config() (faucet.port.Port method), 114
check_config() (faucet.router.Router method), 116
check_config() (faucet.vlan.VLAN method), 140
check_config() (in module faucet.check_faucet_config), 99
check_path() (faucet.faucet_experimental_event.FaucetExperimentalEventNotif method), 107
clear_cache_hosts_on_port() (faucet.vlan.VLAN method), 140
close_logger() (in module faucet.valve_util), 138
close_logs() (faucet.valve.Valve method), 118
combinatorial_port_flood (faucet.dp.DP attribute), 102
Conf (class in faucet.conf), 100
conf (faucet.gauge_influx.InfluxShipper attribute), 111
conf_hash() (faucet.conf.Conf method), 100
config_changed() (in module faucet.config_parser_util), 101
config_file (faucet.valves_manager.ConfigWatcher attribute), 139
config_file_hash() (in module faucet.config_parser_util), 101
config_file_stats (faucet.valves_manager.ConfigWatcher attribute), 139
config_hashes (faucet.valves_manager.ConfigWatcher attribute), 139
configured (faucet.dp.DP attribute), 102
ConfigWatcher (class in faucet.valves_manager), 138
connect_or_disconnect_handler() (faucet.valve_ryuapp.RyuAppBase method), 137

construct_mapping() (faucet.config_parser_util.UniqueKeyLoader method), 101
content_changed() (faucet.valves_manager.ConfigWatcher method), 139
CONTROL_ETH_TYPES (faucet.valve_route.ValveIPv4RouteManager attribute), 134
CONTROL_ETH_TYPES (faucet.valve_route.ValveIPv6RouteManager attribute), 135
CONTROL_ETH_TYPES (faucet.valve_route.ValveRouteManager attribute), 135
control_plane_handler() (faucet.valve_route.ValveIPv4RouteManager method), 134
control_plane_handler() (faucet.valve_route.ValveIPv6RouteManager method), 135
control_plane_handler() (faucet.valve_route.ValveRouteManager method), 136
controller_pps_meteradd() (in module faucet.valve_of), 124
controller_pps_meterdel() (in module faucet.valve_of), 124
cookie (faucet.dp.DP attribute), 102
create_ryu_structure() (faucet.tfm_pipeline.OpenflowToRyuTranslator method), 117

D

datapath_disconnect() (faucet.valve.Valve method), 118
datapath_disconnect() (faucet.valve.Valve method), 118
db (faucet.watcher_conf.WatcherConf attribute), 145
dead() (faucet.valve_route.NextHop method), 134
debug() (faucet.valve.ValveLogger method), 121
dec_ip_ttl() (in module faucet.valve_of), 124
DEC_TTL (faucet.valve.AlliedTelesis attribute), 117
DEC_TTL (faucet.valve.ArubaValve attribute), 117
DEC_TTL (faucet.valve.Valve attribute), 118
decode_value() (in module faucet.fctl), 107
dedupe_ofmsgs() (in module faucet.valve_of), 125
defaults (faucet.acl.ACL attribute), 99
defaults (faucet.conf.Conf attribute), 100
defaults (faucet.dp.DP attribute), 102
defaults (faucet.meter.Meter attribute), 114
defaults (faucet.port.Port attribute), 115
defaults (faucet.router.Router attribute), 116
defaults (faucet.vlan.VLAN attribute), 140
defaults (faucet.watcher_conf.WatcherConf attribute), 145
defaults_types (faucet.acl.ACL attribute), 99
defaults_types (faucet.conf.Conf attribute), 100
defaults_types (faucet.dp.DP attribute), 102
defaults_types (faucet.meter.Meter attribute), 114
defaults_types (faucet.port.Port attribute), 115
defaults_types (faucet.router.Router attribute), 116

- defaults_types (faucet.vlan.VLAN attribute), 140
 - defaults_types (faucet.watcher_conf.WatcherConf attribute), 145
 - del_route() (faucet.valve.Valve method), 118
 - del_route() (faucet.valve_route.ValveRouteManager method), 136
 - del_route() (faucet.vlan.VLAN method), 140
 - delete() (faucet.valve_table.ValveGroupEntry method), 137
 - delete_all() (faucet.valve_table.ValveGroupTable method), 137
 - delete_host_from_vlan() (faucet.valve_host.ValveHostManager method), 124
 - delete_port_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
 - delete_vlan_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
 - desc_stats_reply_handler() (faucet.faucet.Faucet method), 105
 - desc_stats_request() (in module faucet.valve_of), 125
 - description (faucet.port.Port attribute), 115
 - devid_present() (in module faucet.valve_of), 125
 - DP (class in faucet.dp), 101
 - dp (faucet.watcher_conf.WatcherConf attribute), 145
 - dp_acls (faucet.dp.DP attribute), 102
 - dp_config_path() (in module faucet.config_parser_util), 101
 - dp_id (faucet.dp.DP attribute), 102
 - dp_id (faucet.port.Port attribute), 115
 - dp_id (faucet.vlan.VLAN attribute), 140
 - dp_include() (in module faucet.config_parser_util), 101
 - dp_init() (faucet.valve.Valve method), 118
 - dp_parser() (in module faucet.config_parser), 100
 - dpid_log() (in module faucet.valve_util), 138
 - drop_broadcast_source_address (faucet.dp.DP attribute), 102
 - drop_spoofed_faucet_mac (faucet.dp.DP attribute), 102
 - dyn_bgp_neighbor_addresses_by_ip (faucet.vlan.VLAN attribute), 140
 - dyn_bgp_server_addresses_by_ip (faucet.vlan.VLAN attribute), 140
 - dyn_faucet_vips_by_ip (faucet.vlan.VLAN attribute), 140
 - dyn_finalized (faucet.conf.Conf attribute), 100
 - dyn_gws_by_ip (faucet.vlan.VLAN attribute), 140
 - dyn_hash (faucet.conf.Conf attribute), 100
 - dyn_host_cache (faucet.vlan.VLAN attribute), 140
 - dyn_host_cache_by_port (faucet.vlan.VLAN attribute), 141
 - dyn_lacp_up (faucet.port.Port attribute), 115
 - dyn_lacp_updated_time (faucet.port.Port attribute), 115
 - dyn_last_ban_time (faucet.port.Port attribute), 115
 - dyn_last_coldstart_time (faucet.dp.DP attribute), 102
 - dyn_last_lacp_pkt (faucet.port.Port attribute), 115
 - dyn_last_lldp_beacon_time (faucet.port.Port attribute), 115
 - dyn_last_time_hosts_expired (faucet.vlan.VLAN attribute), 141
 - dyn_learn_ban_count (faucet.port.Port attribute), 115
 - dyn_learn_ban_count (faucet.vlan.VLAN attribute), 141
 - dyn_neigh_cache_by_ip (faucet.vlan.VLAN attribute), 141
 - dyn_oldest_host_time (faucet.vlan.VLAN attribute), 141
 - dyn_phys_up (faucet.port.Port attribute), 115
 - dyn_routes_by_ip (faucet.vlan.VLAN attribute), 141
- ## E
- edge_learn_port() (in module faucet.valve_packet), 130
 - edge_learn_port() (faucet.valve_flood.ValveFloodManager method), 122
 - edge_learn_port() (faucet.valve_flood.ValveFloodStackManager method), 123
 - enabled (faucet.port.Port attribute), 115
 - entries (faucet.valve_table.ValveGroupTable attribute), 137
 - entry (faucet.meter.Meter attribute), 114
 - entry_msg (faucet.meter.Meter attribute), 114
 - error() (faucet.valve.ValveLogger method), 121
 - error_handler() (faucet.faucet.Faucet method), 105
 - ETH_TYPE (faucet.valve_route.ValveIPv4RouteManager attribute), 134
 - ETH_TYPE (faucet.valve_route.ValveIPv6RouteManager attribute), 135
 - ETH_TYPE (faucet.valve_route.ValveRouteManager attribute), 135
 - ETH_TYPES_PARSERS (faucet.valve_packet.PacketMeta attribute), 129
 - EventFaucetAdvertise (class in faucet.faucet), 104
 - EventFaucetExperimentalAPIRegistered (class in faucet.faucet), 104
 - EventFaucetLLDPAdvertise (class in faucet.faucet), 104
 - EventFaucetMetricUpdate (class in faucet.faucet), 104
 - EventFaucetResolveGateways (class in faucet.faucet), 104
 - EventFaucetStateExpire (class in faucet.faucet), 104
 - EventReconfigure (class in faucet.valve_ryuapp), 136
 - exact_match (faucet.acl.ACL attribute), 99
 - exc_logname (faucet.faucet.Faucet attribute), 105
 - exc_logname (faucet.gauge.Gauge attribute), 108
 - exc_logname (faucet.valve_ryuapp.RyuAppBase attribute), 137
 - expire_cache_host() (faucet.vlan.VLAN method), 141
 - expire_cache_hosts() (faucet.vlan.VLAN method), 141
 - expire_hosts_from_vlan() (faucet.valve_host.ValveHostFlowRemovedManager method), 123

`expire_hosts_from_vlan()`
(`faucet.valve_host.ValveHostManager`
method), 124

F

`Faucet` (class in `faucet.faucet`), 104
`faucet` (module), 145
`faucet.acl` (module), 98
`faucet.check_faucet_config` (module), 99
`faucet.conf` (module), 100
`faucet.config_parser` (module), 100
`faucet.config_parser_util` (module), 101
`faucet.dp` (module), 101
`faucet.faucet` (module), 104
`faucet.faucet_bgp` (module), 106
`faucet.faucet_experimental_api` (module), 106
`faucet.faucet_experimental_event` (module), 107
`faucet.faucet_metrics` (module), 107
`faucet.faucet_pipeline` (module), 107
`faucet.fctl` (module), 107
`faucet.gauge` (module), 108
`faucet.gauge_influx` (module), 108
`faucet.gauge_pollers` (module), 111
`faucet.gauge_prom` (module), 113
`faucet.meter` (module), 114
`faucet.port` (module), 114
`faucet.prom_client` (module), 116
`faucet.router` (module), 116
`faucet.tfm_pipeline` (module), 117
`faucet.valve` (module), 117
`faucet.valve_acl` (module), 122
`faucet.valve_flood` (module), 122
`faucet.valve_host` (module), 123
`faucet.valve_of` (module), 124
`faucet.valve_of_old` (module), 129
`faucet.valve_packet` (module), 129
`faucet.valve_route` (module), 134
`faucet.valve_ryuapp` (module), 136
`faucet.valve_table` (module), 137
`faucet.valve_util` (module), 138
`faucet.valves_manager` (module), 138
`faucet.vlan` (module), 139
`faucet.watcher` (module), 142
`faucet.watcher_conf` (module), 144
`faucet_async()` (in module `faucet.valve_of`), 125
`faucet_config()` (in module `faucet.valve_of`), 125
`faucet_dp_mac` (`faucet.dp.DP` attribute), 102
`faucet_lldp_tlvs()` (in module `faucet.valve_packet`), 130
`faucet_mac` (`faucet.vlan.VLAN` attribute), 141
`faucet_oui()` (in module `faucet.valve_packet`), 130
`faucet_vips` (`faucet.vlan.VLAN` attribute), 141
`faucet_vips_by_ipv()` (`faucet.vlan.VLAN` method), 141
`FaucetBgp` (class in `faucet.faucet_bgp`), 106

`FaucetExperimentalAPI` (class in `faucet.faucet_experimental_api`), 106
`FaucetExperimentalEventNotifier` (class in `faucet.faucet_experimental_event`), 107
`FaucetMetrics` (class in `faucet.faucet_metrics`), 107
`features_handler()` (`faucet.faucet.Faucet` method), 105
`files_changed()` (`faucet.valves_manager.ConfigWatcher` method), 139
`finalize()` (`faucet.conf.Conf` method), 100
`finalize()` (`faucet.port.Port` method), 115
`finalize_config()` (`faucet.dp.DP` method), 102
`FLOOD_DSTS` (`faucet.valve_flood.ValveFloodManager` attribute), 122
`flood_manager` (`faucet.valve.Valve` attribute), 118
`flood_pkt()` (`faucet.vlan.VLAN` method), 141
`flood_ports()` (`faucet.vlan.VLAN` method), 141
`flood_tagged_port_outputs()` (in module `faucet.valve_of`), 125
`flood_untagged_port_outputs()` (in module `faucet.valve_of`), 125
`flow_timeout()` (`faucet.valve.Valve` method), 118
`flow_timeout()` (`faucet.valve_host.ValveHostFlowRemovedManager` method), 123
`flow_timeout()` (`faucet.valve_host.ValveHostManager` method), 124
`flowcontroller()` (`faucet.valve_table.ValveTable` method), 138
`flowdel()` (`faucet.valve_table.ValveTable` method), 138
`flowdrop()` (`faucet.valve_table.ValveTable` method), 138
`flowmod()` (`faucet.valve_table.ValveTable` method), 138
`flowmod()` (in module `faucet.valve_of`), 125
`flowremoved_handler()` (`faucet.faucet.Faucet` method), 105
`from_connected_to_vip()` (`faucet.vlan.VLAN` method), 141

G

`Gauge` (class in `faucet.gauge`), 108
`GaugeFlowTableInfluxDBLogger` (class in `faucet.gauge_influx`), 108
`GaugeFlowTableLogger` (class in `faucet.watcher`), 142
`GaugeFlowTablePoller` (class in `faucet.gauge_pollers`), 111
`GaugeFlowTablePrometheusPoller` (class in `faucet.gauge_prom`), 113
`GaugePoller` (class in `faucet.gauge_pollers`), 111
`GaugePortStateInfluxDBLogger` (class in `faucet.gauge_influx`), 109
`GaugePortStateLogger` (class in `faucet.watcher`), 143
`GaugePortStatePoller` (class in `faucet.gauge_pollers`), 112
`GaugePortStatePrometheusPoller` (class in `faucet.gauge_prom`), 113
`GaugePortStatsInfluxDBLogger` (class in `faucet.gauge_influx`), 110

GaugePortStatsLogger (class in faucet.watcher), 143
 GaugePortStatsPoller (class in faucet.gauge_pollers), 112
 GaugePortStatsPrometheusPoller (class in faucet.gauge_prom), 114
 GaugePrometheusClient (class in faucet.gauge_prom), 114
 GaugeThreadPoller (class in faucet.gauge_pollers), 112
 get_config() (faucet.faucet.Faucet method), 105
 get_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
 get_config_changes() (faucet.dp.DP method), 102
 get_config_dict() (faucet.dp.DP method), 102
 get_config_dict() (faucet.valve.Valve method), 118
 get_config_for_api() (in module faucet.config_parser), 100
 get_entry() (faucet.valve_table.ValveGroupTable method), 137
 get_logger() (in module faucet.config_parser_util), 101
 get_logger() (in module faucet.valve_util), 138
 get_native_vlan() (faucet.dp.DP method), 102
 get_ports() (faucet.vlan.VLAN method), 141
 get_setting() (faucet.valve_ryuapp.RyuAppBase method), 137
 get_setting() (in module faucet.valve_util), 138
 get_sys_prefix() (in module faucet.valve_util), 138
 get_tables() (faucet.dp.DP method), 102
 get_tables() (faucet.faucet.Faucet method), 105
 get_tables() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
 goto_table() (in module faucet.valve_of), 125
 group_act() (in module faucet.valve_of), 125
 group_flood_buckets() (in module faucet.valve_of), 125
 group_id_from_str() (faucet.valve_table.ValveGroupTable static method), 137
 group_table (faucet.dp.DP attribute), 102
 group_table_routing (faucet.dp.DP attribute), 102
 groupadd() (in module faucet.valve_of), 125
 groupadd_ff() (in module faucet.valve_of), 125
 groupdel() (in module faucet.valve_of), 125
 groupmod() (in module faucet.valve_of), 125
 groupmod_ff() (in module faucet.valve_of), 125
 groups (faucet.dp.DP attribute), 102

H

hairpin (faucet.port.Port attribute), 115
 hairpin_ports() (faucet.vlan.VLAN method), 141
 high_priority (faucet.dp.DP attribute), 103
 host_manager (faucet.valve.Valve attribute), 118
 HostCacheEntry (class in faucet.vlan), 139
 hosts() (faucet.port.Port method), 115
 hosts_count() (faucet.port.Port method), 115
 hosts_count() (faucet.vlan.VLAN method), 141

I

ICMP_TYPE (faucet.valve_route.ValveIPv4RouteManager attribute), 134
 ICMP_TYPE (faucet.valve_route.ValveIPv6RouteManager attribute), 135
 ICMP_TYPE (faucet.valve_route.ValveRouteManager attribute), 135
 icmpv6_echo_reply() (in module faucet.valve_packet), 130
 ignore_learn_ins (faucet.dp.DP attribute), 103
 ignore_port() (in module faucet.valve_of), 125
 ignore_subconf() (faucet.conf.Conf method), 100
 in_port_tables() (faucet.dp.DP method), 103
 InfluxShipper (class in faucet.gauge_influx), 111
 info() (faucet.valve.ValveLogger method), 121
 interface_ranges (faucet.dp.DP attribute), 103
 interfaces (faucet.dp.DP attribute), 103
 InvalidConfigError, 100
 ip_dsts_for_ip_gw() (faucet.vlan.VLAN method), 141
 ip_in_vip_subnet() (faucet.vlan.VLAN method), 141
 ip_ver() (faucet.valve_packet.PacketMeta method), 129
 IPV (faucet.valve_route.ValveIPv4RouteManager attribute), 134
 IPV (faucet.valve_route.ValveIPv6RouteManager attribute), 135
 IPV (faucet.valve_route.ValveRouteManager attribute), 135
 ipv4_parseable() (in module faucet.valve_packet), 131
 ipv6_link_eth_mcast() (in module faucet.valve_packet), 131
 ipv6_solicited_node_from_ucast() (in module faucet.valve_packet), 131
 ipvs() (faucet.vlan.VLAN method), 141
 is_active() (faucet.gauge_pollers.GaugePoller method), 112
 is_active() (faucet.gauge_pollers.GaugeThreadPoller method), 113
 is_delete() (in module faucet.valve_of), 126
 is_faucet_vip() (faucet.vlan.VLAN method), 141
 is_flowdel() (in module faucet.valve_of), 126
 is_flowmod() (in module faucet.valve_of), 126
 is_groupadd() (in module faucet.valve_of), 126
 is_groupdel() (in module faucet.valve_of), 126
 is_groupmod() (in module faucet.valve_of), 126
 is_meteradd() (in module faucet.valve_of), 126
 is_meterdel() (in module faucet.valve_of), 126
 is_metermod() (in module faucet.valve_of), 126
 is_registered() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
 is_table_features_req() (in module faucet.valve_of), 127

K

kill_on_exception() (in module faucet.valve_util), 138

L

`lACP_down()` (faucet.valve.Valve method), 118
`lACP_handler()` (faucet.valve.Valve method), 118
`lACP_reply()` (in module faucet.valve_packet), 131
`lACP_timeout` (faucet.dp.DP attribute), 103
`lACP_up()` (faucet.valve.Valve method), 119
`lags()` (faucet.vlan.VLAN method), 141
`learn_ban_timeout` (faucet.dp.DP attribute), 103
`learn_host_on_vlan_port_flows()`
 (faucet.valve_host.ValveHostManager
 method), 124
`learn_host_on_vlan_ports()`
 (faucet.valve_host.ValveHostManager
 method), 124
`learn_host_timeouts()` (faucet.valve_host.ValveHostFlowRemovedManager
 method), 123
`learn_host_timeouts()` (faucet.valve_host.ValveHostManager
 method), 124
`learn_jitter` (faucet.dp.DP attribute), 103
`lldp_beacon` (faucet.dp.DP attribute), 103
`lldp_beacon` (faucet.port.Port attribute), 115
`lldp_beacon()` (in module faucet.valve_packet), 132
`lldp_beacon_defaults_types` (faucet.dp.DP attribute), 103
`lldp_beacon_defaults_types` (faucet.port.Port attribute),
 115
`lldp_beacon_enabled()` (faucet.port.Port method), 115
`lldp_handler()` (faucet.valve.Valve method), 119
`lldp_org_tlv_defaults_types` (faucet.port.Port attribute),
 115
`load_configs()` (faucet.valves_manager.ValvesManager
 method), 139
`load_tables()` (faucet.tfm_pipeline.LoadRyuTables
 method), 117
`LoadRyuTables` (class in faucet.tfm_pipeline), 117
`logger` (faucet.gauge_influx.InfluxShipper attribute), 111
`logger` (faucet.valve.Valve attribute), 119
`logname` (faucet.faucet.Faucet attribute), 105
`logname` (faucet.gauge.Gauge attribute), 108
`logname` (faucet.valve_ryuapp.RyuAppBase attribute),
 137
`loop_protect` (faucet.port.Port attribute), 115
`low_priority` (faucet.dp.DP attribute), 103

M

`mac_addr_is_unicast()` (in module faucet.valve_packet),
 132
`mac_byte_mask()` (in module faucet.valve_packet), 132
`main()` (in module faucet.check_faucet_config), 99
`main()` (in module faucet.fctl), 107
`make_point()` (faucet.gauge_influx.InfluxShipper static
 method), 111
`make_port_point()` (faucet.gauge_influx.InfluxShipper
 method), 111
`make_wsgi_app()` (in module faucet.prom_client), 116

`match()` (faucet.valve_table.ValveTable method), 138
`match()` (in module faucet.valve_of), 127
`match_from_dict()` (in module faucet.valve_of), 127
`match_tables()` (faucet.dp.DP method), 103
`max_host_fib_retry_count` (faucet.dp.DP attribute), 103
`max_hosts` (faucet.port.Port attribute), 115
`max_hosts` (faucet.vlan.VLAN attribute), 141
`max_hosts_per_resolve_cycle` (faucet.dp.DP attribute),
 103
`MAX_LEN` (faucet.valve_route.ValveRouteManager at-
 tribute), 135
`max_resolve_backoff_time` (faucet.dp.DP attribute), 103
`merge_dyn()` (faucet.conf.Conf method), 100
`Meter` (class in faucet.meter), 114
`meter_id` (faucet.meter.Meter attribute), 114
`meteradd()` (in module faucet.valve_of), 127
`meterdel()` (in module faucet.valve_of), 127
`meters` (faucet.dp.DP attribute), 103
`metric_update()` (faucet.faucet.Faucet method), 105
`metrics` (faucet.faucet.Faucet attribute), 105
`metrics` (faucet.gauge_prom.GaugePrometheusClient at-
 tribute), 114
`metrics_rate_limit_sec` (faucet.dp.DP attribute), 103
`MIN_ETH_TYPE_PKT_SIZE`
 (faucet.valve_packet.PacketMeta attribute),
 129
`mirror` (faucet.port.Port attribute), 115
`mirror_actions()` (faucet.port.Port method), 115
`mirrored_ports()` (faucet.vlan.VLAN method), 141
`modify()` (faucet.valve_table.ValveGroupEntry method),
 137

N

`name` (faucet.dp.DP attribute), 103
`name` (faucet.port.Port attribute), 115
`name` (faucet.vlan.VLAN attribute), 142
`native_vlan` (faucet.port.Port attribute), 115
`nd_advert()` (in module faucet.valve_packet), 132
`nd_request()` (in module faucet.valve_packet), 132
`neigh_cache_by_ipv()` (faucet.vlan.VLAN method), 142
`neigh_cache_count_by_ipv()` (faucet.vlan.VLAN
 method), 142
`new_valve()` (faucet.valves_manager.ValvesManager
 method), 139
`NextHop` (class in faucet.valve_route), 134
`nexthop_dead()` (faucet.valve_route.ValveRouteManager
 method), 136
`no_response()` (faucet.gauge_pollers.GaugeFlowTablePoller
 method), 111
`no_response()` (faucet.gauge_pollers.GaugePoller
 method), 112
`no_response()` (faucet.gauge_pollers.GaugePortStatePoller
 method), 112

- no_response() (faucet.gauge_pollers.GaugePortStatsPoller method), 112
- no_response() (faucet.gauge_pollers.GaugeThreadPoller method), 113
- no_response() (faucet.watcher.GaugePortStateLogger static method), 143
- NonBlockLock (class in faucet.faucet_experimental_event), 107
- notifier (faucet.faucet.Faucet attribute), 105
- notify() (faucet.faucet_experimental_event.FaucetExperimentalEventNotifier method), 107
- number (faucet.port.Port attribute), 115
- ## O
- ofchannel_log() (faucet.valve.Valve method), 119
- ofchannel_logger (faucet.valve.Valve attribute), 119
- ofdescstats_handler() (faucet.valve.Valve method), 119
- oferror() (faucet.valve.Valve method), 119
- OFP_VERSIONS (faucet.valve_ryuapp.RyuAppBase attribute), 136
- op_status_reconf (faucet.port.Port attribute), 115
- OpenflowToRyuTranslator (class in faucet.tfm_pipeline), 117
- output_actions_types (faucet.acl.ACL attribute), 99
- output_controller() (in module faucet.valve_of), 127
- output_in_port() (in module faucet.valve_of), 127
- output_only (faucet.port.Port attribute), 115
- output_only_ports (faucet.dp.DP attribute), 103
- output_port() (faucet.vlan.VLAN method), 142
- output_port() (in module faucet.valve_of), 127
- override_output_port (faucet.port.Port attribute), 115
- OVSValve (class in faucet.valve), 117
- ## P
- packet_complete() (faucet.valve_packet.PacketMeta method), 129
- packet_in_handler() (faucet.faucet.Faucet method), 105
- packetin_pps (faucet.dp.DP attribute), 103
- PacketMeta (class in faucet.valve_packet), 129
- packetout() (in module faucet.valve_of), 127
- parse_args() (in module faucet.fctl), 107
- parse_configs() (faucet.valves_manager.ValvesManager method), 139
- parse_eth_pkt() (in module faucet.valve_packet), 133
- parse_lacp_pkt() (in module faucet.valve_packet), 133
- parse_lldp() (in module faucet.valve_packet), 133
- parse_packet_in_pkt() (in module faucet.valve_packet), 133
- parse_pkt_meta() (faucet.valve.Valve method), 119
- parse_rcv_packet() (faucet.valve.Valve method), 119
- parse_vlan_pkt() (in module faucet.valve_packet), 133
- peer_stack_up_ports() (faucet.dp.DP method), 103
- permanent_learn (faucet.port.Port attribute), 115
- PIPELINE_CONF (faucet.valve.ArubaValve attribute), 117
- PIPELINE_CONF (faucet.valve.TfmValve attribute), 117
- pipeline_config_dir (faucet.dp.DP attribute), 103
- pkt_out_port() (faucet.vlan.VLAN method), 142
- pop_vlan() (in module faucet.valve_of), 128
- Port (class in faucet.port), 114
- port_add() (faucet.valve.Valve method), 119
- port_delete() (faucet.valve.Valve method), 120
- port_delete() (faucet.vlan.VLAN method), 142
- port_is_untagged() (faucet.vlan.VLAN method), 142
- port_no_valid() (faucet.valve.Valve method), 120
- port_status_from_state() (in module faucet.valve_of), 128
- port_status_handler() (faucet.faucet.Faucet method), 105
- port_status_handler() (faucet.valve.Valve method), 120
- ports (faucet.dp.DP attribute), 103
- ports_add() (faucet.valve.Valve method), 120
- ports_delete() (faucet.valve.Valve method), 120
- prepare_send_flows() (faucet.valve.Valve method), 120
- priority_offset (faucet.dp.DP attribute), 103
- proactive_arp_limit (faucet.vlan.VLAN attribute), 142
- proactive_learn (faucet.dp.DP attribute), 103
- proactive_nd_limit (faucet.vlan.VLAN attribute), 142
- prom_client (faucet.gauge.Gauge attribute), 108
- prom_client (faucet.watcher_conf.WatcherConf attribute), 145
- PromClient (class in faucet.prom_client), 116
- push_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
- push_vlan() (in module faucet.valve_acl), 122
- push_vlan_act() (in module faucet.valve_of), 128
- ## R
- rate_limit_packet_ins() (faucet.valve.Valve method), 120
- rcv_packet() (faucet.valve.Valve method), 120
- read_config() (in module faucet.config_parser_util), 101
- receive_lldp (faucet.port.Port attribute), 115
- recent_ofmsgs (faucet.valve.Valve attribute), 120
- reconnect_handler() (faucet.valve_ryuapp.RyuAppBase method), 137
- release() (faucet.faucet_experimental_event.NonBlockLock method), 107
- reload_config() (faucet.faucet.Faucet method), 105
- reload_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 106
- reload_config() (faucet.gauge.Gauge method), 108
- reload_config() (faucet.valve.Valve method), 120
- reload_config() (faucet.valve_ryuapp.RyuAppBase method), 137
- reparse() (faucet.valve_packet.PacketMeta method), 129
- reparse_all() (faucet.valve_packet.PacketMeta method), 129
- reparse_ip() (faucet.valve_packet.PacketMeta method), 129

report_dp_status() (faucet.gauge_pollers.GaugePoller method), 112
report_label_match_metrics() (in module faucet.fctl), 108
request_reload_configs() (faucet.valves_manager.ValvesManager method), 139
REQUIRED_LABELS (faucet.prom_client.PromClient attribute), 116
reregister_flow_vars() (faucet.gauge_prom.GaugePrometheusClient method), 114
reset() (faucet.faucet_bgp.FaucetBgp method), 106
reset_caches() (faucet.vlan.VLAN method), 142
reset_dpuid() (faucet.faucet_metrics.FaucetMetrics method), 107
reset_ports() (faucet.vlan.VLAN method), 142
reset_refs() (faucet.dp.DP method), 103
resolve_gateways() (faucet.valve.Valve method), 121
resolve_gateways() (faucet.valve_route.ValveRouteManagerset_vlan_vid() (in module faucet.valve_of), 128
resolve_gw_on_port() (faucet.valve_route.ValveIPv4RouteManager method), 134
resolve_gw_on_port() (faucet.valve_route.ValveIPv6RouteManager method), 135
resolve_gw_on_port() (faucet.valve_route.ValveRouteManagerset_vlan_vid() (in module faucet.valve_of), 128
resolve_gw_on_vlan() (faucet.valve_route.ValveIPv4RouteManager method), 134
resolve_gw_on_vlan() (faucet.valve_route.ValveIPv6RouteManager method), 135
resolve_gw_on_vlan() (faucet.valve_route.ValveRouteManagerset_vlan_vid() (in module faucet.valve_of), 128
resolve_stack_topology() (faucet.dp.DP method), 103
rewrite_vlan() (in module faucet.valve_acl), 122
route_count_by_ipv() (faucet.vlan.VLAN method), 142
Router (class in faucet.router), 116
router_advert() (in module faucet.valve_packet), 133
routers (faucet.dp.DP attribute), 103
routes (faucet.vlan.VLAN attribute), 142
routes_by_ipv() (faucet.vlan.VLAN method), 142
rule_types (faucet.acl.ACL attribute), 99
rules (faucet.acl.ACL attribute), 99
running (faucet.dp.DP attribute), 103
running() (faucet.gauge_pollers.GaugePoller method), 112
running() (faucet.port.Port method), 116
RyuAppBase (class in faucet.valve_ryuapp), 136

S

scrape_prometheus() (in module faucet.fctl), 108
send_flows() (faucet.valve.Valve method), 121
send_lldp Beacons() (faucet.valve.Valve method), 121
send_req() (faucet.gauge_pollers.GaugeFlowTablePoller method), 111
send_req() (faucet.gauge_pollers.GaugePoller method), 112
send_req() (faucet.gauge_pollers.GaugePortStatePoller method), 112
send_req() (faucet.gauge_pollers.GaugePortStatsPoller method), 112
send_req() (faucet.gauge_pollers.GaugeThreadPoller method), 113
send_req() (faucet.watcher.GaugePortStateLogger static method), 143
server (faucet.prom_client.PromClient attribute), 116
set_defaults() (faucet.conf.Conf method), 100
set_defaults() (faucet.dp.DP method), 103
set_defaults() (faucet.port.Port method), 116
set_defaults() (faucet.vlan.VLAN method), 142
set_eth_dst() (in module faucet.valve_of), 128
set_eth_src() (in module faucet.valve_of), 128
set_field() (in module faucet.valve_of), 128
set_vlan_vid() (in module faucet.valve_of), 128
ship_error_prefix (faucet.gauge_influx.InfluxShipper attribute), 111
ship_points() (faucet.gauge_influx.InfluxShipper method), 111
shortest_path() (faucet.dp.DP method), 103
shortest_path_port() (faucet.dp.DP method), 103
shortest_path_to_root() (faucet.dp.DP method), 104
start_bgp_speakers() (faucet.faucet_bgp.FaucetBgp method), 106
MatchHandler() (faucet.valve_ryuapp.RyuAppBase method), 137
SKIP_VALIDATION_TABLES (faucet.valve.TfmValve attribute), 117
stack (faucet.dp.DP attribute), 104
stack (faucet.port.Port attribute), 116
stack_defaults_types (faucet.dp.DP attribute), 104
stack_defaults_types (faucet.port.Port attribute), 116
stack_ports (faucet.dp.DP attribute), 104
start() (faucet.faucet.Faucet method), 105
start() (faucet.faucet_experimental_event.FaucetExperimentalEventNotifier method), 107
start() (faucet.gauge_pollers.GaugePoller method), 112
start() (faucet.gauge_pollers.GaugeThreadPoller method), 113
start() (faucet.prom_client.PromClient method), 116
start() (faucet.valve_ryuapp.RyuAppBase method), 137
stat_config_files() (in module faucet.valve_util), 138
state_expire() (faucet.valve.Valve method), 121
stop() (faucet.gauge_pollers.GaugePoller method), 112
stop() (faucet.gauge_pollers.GaugeThreadPoller method), 113
switch_features() (faucet.valve.TfmValve method), 117
switch_features() (faucet.valve.Valve method), 121

T

table_features() (in module faucet.valve_of), 128

- table_tags (faucet.gauge_prom.GaugeFlowTablePrometheus attribute), 113
- tables (faucet.dp.DP attribute), 104
- tables_by_id (faucet.dp.DP attribute), 104
- tagged (faucet.vlan.VLAN attribute), 142
- tagged_flood_ports() (faucet.vlan.VLAN method), 142
- tagged_vlans (faucet.port.Port attribute), 116
- targeted_gw_resolution (faucet.vlan.VLAN attribute), 142
- test_config_condition() (in module faucet.conf), 100
- TfmValve (class in faucet.valve), 117
- thread (faucet.prom_client.PromClient attribute), 116
- timeout (faucet.dp.DP attribute), 104
- to_conf() (faucet.acl.ACL method), 99
- to_conf() (faucet.conf.Conf method), 100
- to_conf() (faucet.dp.DP method), 104
- to_conf() (faucet.port.Port method), 116
- to_conf() (faucet.vlan.VLAN method), 142
- ## U
- unicast_flood (faucet.port.Port attribute), 116
- unicast_flood (faucet.vlan.VLAN attribute), 142
- UniqueKeyLoader (class in faucet.config_parser_util), 101
- untagged (faucet.vlan.VLAN attribute), 142
- untagged_flood_ports() (faucet.vlan.VLAN method), 142
- update() (faucet.conf.Conf method), 100
- update() (faucet.gauge_influx.GaugeFlowTableInfluxDBLogger method), 109
- update() (faucet.gauge_influx.GaugePortStateInfluxDBLogger method), 110
- update() (faucet.gauge_influx.GaugePortStatsInfluxDBLogger method), 111
- update() (faucet.gauge_pollers.GaugePoller method), 112
- update() (faucet.gauge_prom.GaugeFlowTablePrometheusPoller method), 113
- update() (faucet.gauge_prom.GaugePortStatePrometheusPoller method), 113
- update() (faucet.gauge_prom.GaugePortStatsPrometheusPoller method), 114
- update() (faucet.valves_manager.ConfigWatcher method), 139
- update() (faucet.watcher.GaugeFlowTableLogger method), 143
- update() (faucet.watcher.GaugePortStateLogger method), 143
- update() (faucet.watcher.GaugePortStatsLogger method), 143
- update_buckets() (faucet.valve_table.ValveGroupEntry method), 137
- update_config_metrics() (faucet.valve.Valve method), 121
- update_metrics() (faucet.faucet_bgp.FaucetBgp method), 106
- update_metrics() (faucet.valve.Valve method), 121
- update_metrics() (faucet.valves_manager.ValvesManager method), 139
- update_watcher_handler() (faucet.gauge.Gauge method), 108
- USE_BARRIERS (faucet.valve.OVSVValve attribute), 117
- USE_BARRIERS (faucet.valve.Valve attribute), 118
- use_idle_timeout (faucet.dp.DP attribute), 104
- ## V
- Valve (class in faucet.valve), 117
- valve_factory() (in module faucet.valve), 122
- valve_flow_services() (faucet.valves_manager.ValvesManager method), 139
- valve_flowreorder() (in module faucet.valve_of), 128
- valve_match_vid() (in module faucet.valve_of), 128
- valve_packet_in() (faucet.valves_manager.ValvesManager method), 139
- ValveFloodManager (class in faucet.valve_flood), 122
- ValveFloodStackManager (class in faucet.valve_flood), 122
- ValveGroupEntry (class in faucet.valve_table), 137
- ValveGroupTable (class in faucet.valve_table), 137
- ValveHostFlowRemovedManager (class in faucet.valve_host), 123
- ValveHostManager (class in faucet.valve_host), 123
- ValveIPv4RouteManager (class in faucet.valve_route), 134
- ValveIPv6RouteManager (class in faucet.valve_route), 134
- ValveLogger (class in faucet.valve), 121
- ValveRouteManager (class in faucet.valve_route), 135
- valves (faucet.valves_manager.ValvesManager attribute), 139
- valves_manager (faucet.faucet.Faucet attribute), 106
- ValvesManager (class in faucet.valves_manager), 139
- ValveTable (class in faucet.valve_table), 138
- vid (faucet.vlan.VLAN attribute), 142
- vid_present() (in module faucet.valve_of), 128
- vid_valid() (faucet.vlan.VLAN static method), 142
- VLAN (class in faucet.vlan), 139
- vlan_match_tables() (faucet.dp.DP method), 104
- vlans (faucet.dp.DP attribute), 104
- vlans (faucet.router.Router attribute), 116
- vlans() (faucet.port.Port method), 116
- ## W
- warning() (faucet.valve.ValveLogger method), 122
- watcher_factory() (in module faucet.watcher), 144
- watcher_parser() (in module faucet.config_parser), 100
- WatcherConf (class in faucet.watcher_conf), 144
- wildcard_table (faucet.dp.DP attribute), 104