Faucet Documentation

Faucet Developers

Feb 21, 2020
# CONTENTS

1 User Documentation
   1.1 Introduction to Faucet .................................................. 1
   1.2 Tutorials ................................................................. 5
   1.3 Installation ............................................................. 63
   1.4 Configuration ......................................................... 70
   1.5 Monitoring ............................................................. 90
   1.6 Configuration Recipe Book .......................................... 92
   1.7 Vendor-specific Documentation ..................................... 92
   1.8 External Resources ............................................... 124

2 Developer Documentation ............................................... 125
   2.1 Developer Guide .................................................... 125
   2.2 Architecture ......................................................... 130
   2.3 Testing ............................................................... 134
   2.4 Fuzzing ............................................................... 139
   2.5 Source Code .......................................................... 140

3 Indices and tables ...................................................... 209

Python Module Index ..................................................... 211

Index ........................................................................ 213
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.
Faucet Documentation

**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-afl). 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 Release Notes**

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:

<table>
<thead>
<tr>
<th>Old path</th>
<th>New path</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/ryu/faucet</td>
<td>/etc/faucet</td>
</tr>
<tr>
<td>/var/log/ryu/faucet</td>
<td>/var/log/faucet</td>
</tr>
</tbody>
</table>

   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 Installation with 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:

\[
\text{dps:} \\
\text{  mydp:} \\
\text{    combinatorial_port_flood: True}
\]

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 (0xffff) is set.

1.9.0 Release Notes

There are some changes in version 1.9.0 of faucet that may affect how you use it. Below are the changes and how they might affect you.

1. Removing support for older python versions

Starting from faucet 1.9.0 and onwards, faucet now requires a version of python 3.5 or newer to function.

Most currently supported distributions of linux should have a version of python that is compatible, with the notable exception of Debian Jessie which is no longer supported by faucet.

2. Change BGP configuration syntax

Previously, BGP configuration for faucet was attached to a VLAN, for example:

1.1. Introduction to Faucet
As BGP peering in faucet now has the ability to resolve next hops in all VLANs, we have elected to move where BGP is configured.

We have now implemented a new `bgp` router type that can be configured in faucet, similar to how inter-VLAN routing works today, for example this is an example of the new syntax showing how we would convert the configuration shown above:

<table>
<thead>
<tr>
<th>Listing 2: Newer style bgp configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlans:</td>
</tr>
<tr>
<td>internet:</td>
</tr>
<tr>
<td>description: 'internet peering'</td>
</tr>
<tr>
<td>vid: 200</td>
</tr>
<tr>
<td>routers:</td>
</tr>
<tr>
<td>internet-router:</td>
</tr>
<tr>
<td>bgp:</td>
</tr>
<tr>
<td>vlan: internet</td>
</tr>
<tr>
<td>routerid: '127.0.0.2'</td>
</tr>
<tr>
<td>as: 14031</td>
</tr>
<tr>
<td>neighbor_as: 14031</td>
</tr>
<tr>
<td>neighbor_addresses: ['127.0.0.1', '::1']</td>
</tr>
<tr>
<td>server_addresses: ['127.0.0.2', '::1']</td>
</tr>
<tr>
<td>port: 9179</td>
</tr>
<tr>
<td>connect_mode: 'passive'</td>
</tr>
</tbody>
</table>

It is also possible to combine inter-VLAN routing and bgp routing in a single routing instance:

<table>
<thead>
<tr>
<th>Listing 3: Newer style bgp configuration (with IVR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlans:</td>
</tr>
<tr>
<td>office:</td>
</tr>
<tr>
<td>description: 'internet peering'</td>
</tr>
<tr>
<td>vid: 100</td>
</tr>
<tr>
<td>internet:</td>
</tr>
<tr>
<td>description: 'internet peering'</td>
</tr>
<tr>
<td>vid: 200</td>
</tr>
<tr>
<td>routers:</td>
</tr>
<tr>
<td>office-internet-router:</td>
</tr>
<tr>
<td>vlans: [office, internet]</td>
</tr>
<tr>
<td>bgp:</td>
</tr>
<tr>
<td>vlan: internet</td>
</tr>
</tbody>
</table>

(continues on next page)
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 FAUCET dev log 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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet</td>
<td>Network controller</td>
</tr>
<tr>
<td>gauge</td>
<td>Monitoring controller</td>
</tr>
<tr>
<td>prometheus</td>
<td>Monitoring system &amp; time series database</td>
</tr>
<tr>
<td>grafana</td>
<td>Monitoring dashboard</td>
</tr>
</tbody>
</table>

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:

```yaml
# 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"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scape_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:

   ```
   # 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 its 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:

   ```yaml
   vlans:
     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.

   ```bash
   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:

   ```json
   [{
     'advertise_interval': 30,
     'arp_neighbor_timeout': 30,
     'cache_update_guard_time': 150,
     'combinatorial_port_flood': False,
     'cookie': 1524372928,
     'description': 'sw1',
     'dot1x': None,
     'dp_acls': None,
     'dp_id': 1,
     'drop_broadcast_source_address': True,
     'drop_spoofed_faucet_mac': True,
   }
   ```

   (continues on next page)
{'egress_pipeline': False, 'fast_advertise_interval': 5, 'faucet_dp_mac': '0e:00:00:00:00:01', 'global_vlan': 0, 'group_table': False, 'hardware': 'Open vSwitch', 'high_priority': 9001, 'highest_priority': 9099, 'idle_dst': True, 'ignore_learn_ins': 10, 'interface_ranges': OrderedDict(), 'interfaces': {'host1': {'acl_in': None, 'acls_in': None, 'description': 'host1 network namespace', 'dot1x': False, 'enabled': True, 'hairpin': False, 'hairpin_unicast': False, 'lacp': 0, 'lacp_active': False, 'lldp_beacon': OrderedDict(), 'loop_protect': False, 'loop_protect_external': False, 'max_hosts': 255, 'max_lldp_lost': 3, 'mirror': None, 'native_vlan': 'office', 'number': 1, 'opstatus_reconf': True, 'output_only': False, 'permanent_learn': False, 'receive_lldp': False, 'stack': OrderedDict(), 'tagged_vlans': [], 'unicast_flood': True}, 'host2': {'acl_in': None, 'acls_in': None, 'description': 'host2 network namespace', 'dot1x': False, 'enabled': True, 'hairpin': False, 'hairpin_unicast': False, 'lacp': 0, 'lacp_active': False, 'lldp_beacon': OrderedDict(), 'loop_protect': False, 'loop_protect_external': False, 'max_hosts': 255, 'max_lldp_lost': 3, 'mirror': None, 'native_vlan': 'office', 'number': 2, 'opstatus_reconf': True, 'output_only': False, 'permanent_learn': False, 'receive_lldp': False, 'stack': OrderedDict()}, ...
Faucet Documentation

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).

```bash
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 7: /var/log/faucet/faucet.log

<table>
<thead>
<tr>
<th>Log</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet INFO</td>
<td>Loaded configuration from /etc/faucet/faucet.yaml</td>
</tr>
<tr>
<td>faucet INFO</td>
<td>Add new datapath DPID 1 (0x1)</td>
</tr>
<tr>
<td>faucet INFO</td>
<td>Add new datapath DPID 2 (0x2)</td>
</tr>
<tr>
<td>faucet INFO</td>
<td>Configuration /etc/faucet/faucet.yaml changed, analyzing differences</td>
</tr>
<tr>
<td>faucet INFO</td>
<td>Reconfiguring existing datapath DPID 1 (0x1)</td>
</tr>
<tr>
<td>faucet.valve INFO</td>
<td>DPID 1 (0x1) skipping configuration because datapath not up</td>
</tr>
<tr>
<td>faucet INFO</td>
<td>Deleting de-configured DPID 2 (0x2)</td>
</tr>
</tbody>
</table>

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 8: gauge.yaml

```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
    path: 'flow_tables'
  influx:
    type: 'influx'
    influx_db: 'faucet'
    influx_host: 'influxdb'
```

(continues on next page)
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:

```bash
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).

   ```bash
   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

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 copying and pasting the following definitions into our bash terminal:

   ```bash
   # Run command inside network namespace
   as_ns () {
       NAME=$1
       NETNS=faucet-$NAME
       shift
       sudo ip netns exec $NETNS $@
   }
   ```

   ```bash
   sudo apt-get install openvswitch-switch
   ```
# Create network namespace

```bash
create_ns() {  
  NAME=$1  
  IP=$2  
  NETNS=faucet-$1
  sudo ip netns add $NETNS  
  sudo ip link add dev veth-$1 type veth peer name veth0 netns $NETNS  
  sudo ip link set dev veth-$1 up  
  as_ns $1 ip link set dev lo up  
  [ -n "$IP" ] && as_ns $1 ip addr add dev veth0 $IP  
  as_ns $1 ip link set dev veth0 up
}
```

NOTE: all the tutorial helper functions can be defined by sourcing `helper-funcs` into your shell environment.

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

```bash
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:

```bash
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  
-- 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:

```bash
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:
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 --server --pidfile /run/iperf3-host1.pid --daemon
as_ns host2 iperf3 --client 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`.

```bash
# Run command inside network namespace
as_ns () {
    NAME=$1
    NETNS=faucet-$NAME
    shift
    sudo ip netns exec $NETNS $@
}

# Create network namespace
create_ns () {
    NAME=$1
}
```

(continues on next page)
Faucet Documentation

(continued from previous page)

```bash
IP=$2
NETNS=faucet-$\{NAME\}
sudo ip netns add $\{NETNS\}
sudo ip link add dev veth-$\{NAME\} type veth peer name veth0 netns $\{NETNS\}
sudo ip link set dev veth-$\{NAME\} up
as_ns $\{NAME\} ip link set dev lo up
[ -n "$\{IP\}" ] && as_ns $\{NAME\} ip addr add dev veth0 $\{IP\}
as_ns $\{NAME\} ip link set dev veth0 up
```

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

```bash
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 \
   -- 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:

```yaml
Listing 10: /etc/faucet/faucet.yaml

vlans:
  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
```

1.2. Tutorials
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. The first rule that matches a packet, will set the actions for the packet. 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. If no matches are specified, the rule will match all packets.

Actions are used to control what the packet does, for example normal L2 forwarding (‘allow’), apply a ‘meter’ to rate limit traffic, and manipulation of the packet contents and output destination. The 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. The default ‘allow’ for ACLs is False (i.e. drop the packet). ACL rules will need to define ‘allow: True’ for those packets that are to be forwarded.

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’.
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
```

Pings to/from host3 should now fail:

```
as_ns host1 ping 192.168.0.3
```

But the other three hosts should be fine:

```
as_ns host1 ping 192.168.0.2
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
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
13:24:37.857024 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
13:24:40.881129 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.
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 14: /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).
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 15: /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
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’.

```bash
# Run command inside network namespace
as_ns () {
    NAME=$1
    NETNS=faucet-$NAME
    shift
    sudo ip netns exec $NETNS @
}

# Create network namespace
create_ns () {
    NAME=$1
    IP=$2
    NETNS=faucet-$NAME
    sudo ip netns add $NETNS
    sudo ip link add dev veth-$NAME type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth0 up
    as_ns $NAME
    ip link set dev lo up
    [ -n "$IP" ] && as_ns $NAME ip addr add dev veth0 $IP
    as_ns $NAME ip link set dev veth0 up
}

# Clean up namespaces, bridges and processes created during faucet tutorial
cleanup () {
    for NETNS in $(sudo ip netns list | grep "faucet-" | awk '{print $1}'); do
        [ -n "$NETNS" ] || continue
        NAME=${NETNS#faucet-}
        if [ -f "/run/dhclient-$NAME.pid" ]; then
            # Stop dhclient
            sudo pkill -F "/run/dhclient-$NAME.pid"
        fi
        if [ -f "/run/iperf3-$NAME.pid" ]; then
            # Stop iperf3
            sudo pkill -F "/run/iperf3-$NAME.pid"
        fi
        if [ -f "/run/bird-$NAME.pid" ]; then
            # Stop bird
            sudo pkill -F "/run/bird-$NAME.pid"
        fi
        # Remove netns and veth pair
    done
}
```

(continues on next page)
Overview

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

- Use VLANs to segment traffic.
- Create VLAN Trunks.
- Apply an ACL to an entire 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. Mixed - where both native and tagged packets appear on the same port.

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 this diagram nearby 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 \
  -- set bridge br0 fail_mode=secure \
```
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.

```plaintext
create_ns host3 0.0.0.0
create_ns host4 0.0.0.0
add_tagged_interface host3 100 192.168.0.3/24
add_tagged_interface host4 100 192.168.0.4/24
```

- Assign host5 and host6 an IP address from the VLAN 200 range.

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

- Assign host7 and host8 a VLAN interface (vid:300) as they are on a tagged port.

```plaintext
create_ns host7 0.0.0.0
create_ns host8 0.0.0.0
add_tagged_interface host7 300 192.168.3.7/24
add_tagged_interface host8 300 192.168.3.8/24
```

- Add host9 to all VLANs (100, 200, 300) to work as a NFV host.

```plaintext
create_ns host9 0.0.0.0
add_tagged_interface host9 100 192.168.0.9/24
add_tagged_interface host9 200 192.168.2.9/24
add_tagged_interface host9 300 192.168.3.9/24
```

Then connect all the hosts to the switch (br0)

```plaintext
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 16: /etc/faucet/faucet.yaml

```
vlans:
  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: vlan100
      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 (all should work)
2. Ping between hosts in the same VLAN where the one host is native and the other is tagged should work also. In particular between host1 (native VLAN 100) to host3 (tagged VLAN 100).

3. Ping between hosts in different VLANs should fail. To test that let’s add the IP address 192.168.0.5 to host5 (native VLAN 200) and try to ping it from host1 (native VLAN 100).

4. Now we can test the trunk link to host9 from different VLANs (all should work)

**VLAN ACL**

Let’s apply an ACL on a particular VLAN (e.g. VLAN 300). We will block any ICMP packets on VLAN 300. First create an ACL to block the ping. Open /etc/faucet/faucet.yaml and add the `acls` section.

```
Listing 17: /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 VLAN 300.

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

vlans:
  vlan100:
    vid: 100
  vlan200:
    vid: 200
  vlan300:
    vid: 300
  acls_in: [block-ping] # Apply ACL only on vlan300
```
Just before we reload the configuration file. Let’s verify that pinging is working between hosts in VLAN 300.

```
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, etc)

#### 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’.

```bash
# Run command inside network namespace
as_ns () {
    NAME=$1
    NETNS=faucet-$NAME
    shift
    sudo ip netns exec $NETNS @
}

# Create network namespace
create_ns () {
    NAME=$1
    IP=$2
    NETNS=faucet-$NAME
    sudo ip netns add $NETNS
    sudo ip link add dev veth-$NAME type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-$NAME up
    as_ns $NAME ip link set dev lo up
    [ -n "$IP" ] && as_ns $NAME ip addr add dev veth0 $IP
    as_ns $NAME ip link set dev veth0 up
}
```

1.2. Tutorials
• 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 br0
-- set bridge br0 other-config:datapath-id=0000000000000001
-- set bridge br0 other-config:disable-in-band=true
-- set bridge br0 fail_mode=secure
```
In this section we will be using faucet as a gateway for our two hosts and using faucet to route between them. To do this we are going to need to give faucet an IP address on the network. This is accomplished with by using two new options that we haven’t seen before:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet_vips</td>
<td>The IP address for Faucet’s routing interface on this VLAN. Multiple IP addresses (IPv4 &amp; IPv6) can be used.</td>
</tr>
<tr>
<td>faucet_mac</td>
<td>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.</td>
</tr>
</tbody>
</table>

Let’s add the following faucet configuration which makes use of these options.

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

vlans:
  vlan100:
    vid: 100
    faucet_vips: ["10.0.0.254/24"] # Faucet’s virtual IP address for vlan100
    faucet_mac: "00:00:00:00:00:11"
  vlan200:
    vid: 200
    faucet_vips: ["10.0.1.254/24"] # Faucet’s virtual IP address for vlan200
    faucet_mac: "00:00:00:00:00:22"

dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: vlan100
```

(continues on next page)
Now let's signal faucet to reload the configuration file.

```bash
sudo systemctl reload faucet
```

Add a default route on each host to set the gateway to the value we used for `faucet_vips` above.

```bash
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
```

By default traffic between our two hosts will be dropped since they are in different VLANs with different subnets. We can show that by doing the following:

```bash
as_ns host1 ping 10.0.1.2
```

We can change this by enabling inter-VLAN routing between these two VLANs. In faucet you do this by creating a router and specifying which VLANs can route between each other.

In our case we want to enable routing between VLAN 100 and VLAN 200 so we add the following to our configuration file.

```yaml
Listing 20: /etc/faucet/faucet.yaml

routers:
  router-1:
    name: "host2"
    description: "host2 network namespace"
    native_vlan: vlan200

vlans: [vlan100, vlan200]
```

Reload faucet to enable inter-VLAN routing.

```bash
sudo systemctl reload faucet
```

Our ping before from host1 to host2 should now work (the first few packets may get lost as faucet needs to resolve the MAC address of the next hop with ARP).

```bash
as_ns host1 ping 10.0.1.2
```

Inter-VLAN routing by default will allow all traffic to pass between VLANs, if we wanted to change this and restrict communication to a few different IP addresses or TCP/UDP ports, we could apply a VLAN ACL to each VLAN to limit the types of traffic that may pass and what should be dropped.

### Static routing

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

Run the cleanup script to remove old namespaces and switches.

```bash
cleanup
```

Create 3 hosts, in 2 different subnets:

```bash
create_ns host1 10.0.0.1/24
create_ns host2 10.0.0.2/24
create_ns server 10.0.1.1/24
```
Add a default route for each host to the gateway which is faucet's virtual IP address.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_ns host1 ip route add default via 10.0.0.254</td>
</tr>
<tr>
<td>as_ns host2 ip route add default via 10.0.0.254</td>
</tr>
<tr>
<td>as_ns server ip route add default via 10.0.1.254</td>
</tr>
</tbody>
</table>

Create the bridge and add host1, host2 and the server to br0.

```
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-server -- set interface veth-server ofport_request=3 \ 
  -- set-controller br0 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 one for hosts and one for servers. We will also note that inside the configuration for the servers VLAN we see a static route that routes the subnet 192.0.2.0/24 to the server namespace (10.0.1.1).

```
vlans:

  hosts:
    vid: 100
    description: "vlan for clients"
    faucet_mac: "00:00:00:00:00:11"
    faucet_vips: ["10.0.0.254/24"]

  servers:
    vid: 200
    description: "vlan for servers"
    faucet_mac: "00:00:00:00:00:22"
    faucet_vips: ["10.0.1.254/24"]
    routes:
      - route:
          ip_dst: "192.0.2.0/24"
```

(continues on next page)
ip_gw: '10.0.1.1'

routers:
    router-hosts-servers:
        vlans: [hosts, servers]
dps:
    br0:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: hosts
            2:
                name: "host2"
                description: "host2 network namespace"
                native_vlan: hosts
            3:
                name: "server"
                description: "server network namespace"
                native_vlan: servers

Reload Faucet to apply the new configuration.

```bash
sudo systemctl reload faucet
```

We can verify the inter-VLAN Routing is working by pinging the IP address of the server namespace:

```bash
as_ns host1 ping 10.0.1.1
```

We also need to add an additional IP alias to server to test the static route works.

```bash
as_ns server ip address add 192.0.2.1/24 dev veth0
```

And we should now be able to ping our IP alias.

```bash
as_ns host1 ping 192.0.2.1
```

**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

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.

Remove the following lines from `/etc/faucet/faucet.yaml` to remove the static route from faucet:

```yaml
Listing 22: /etc/faucet/faucet.yaml
routes:
- route:
  ip_dst: "192.0.2.0/24"
  ip_gw: '10.0.1.1'
```

Reload Faucet

```
sudo systemctl reload faucet
```

Verify that we can no longer ping the address we were previously static routing.

```
as_ns host1 ping 192.0.2.1
```

Let’s add a new network namespace to run BIRD

```
create_ns bgp 10.0.1.2/24
sudo ovs-vsctl add-port br0 veth-bgp -- set interface veth-bgp ofport_request=4
```

Next we will add a dataplane connection for Faucet so that it can communicate with BIRD running on the bgp namespace.
Faucet Documentation

```
sudo ip link add veth-faucet type veth peer name veth-faucet-ovs
sudo ovs-vsctl add-port br0 veth-faucet-ovs -- set interface veth-faucet-ovs ofport_request=5
sudo ip addr add 10.0.1.3/24 dev veth-faucet
sudo ip link set veth-faucet up
sudo ip link set veth-faucet-ovs up
```

Now install BIRD on the system and stop it from running:

```
sudo apt-get install bird
sudo systemctl stop bird
sudo systemctl stop bird6
```

To configure BIRD add the following to `/etc/bird/bird.conf`, this will create a simple routing setup where BIRD originates a static route for 192.0.2.0/24 and sends this to faucet over BGP.

```
Listing 23: /etc/bird/bird.conf

protocol kernel {
    scan time 60;
    import none;
}
protocol device {
    scan time 60;
}
# Generate static route inside bird
protocol static {
    route 192.0.2.0/24 via 10.0.1.1;
}
# BGP peer with faucet
# Import all routes and export our static route
protocol bgp faucet {
    local as 65001;
    neighbor 10.0.1.3 port 9179 as 65000;
    export all;
    import all;
}
```

We can now start BIRD inside the bgp namespace:

```
as_ns bgp bird -P /run/bird-bgp.pid
```

We’ll configure Faucet to talk to BIRD by adding BGP configuration to `/etc/faucet/faucet.yaml`. Add the following to the routers section.

```
Listing 24: /etc/faucet/faucet.yaml

routers:
    ...  
    bird:
        bgp:
            vlan: servers  # The VLAN faucet use for BGP
            as: 65000     # Faucet’s AS number
            port: 9179   # BGP port for Faucet to listen on.
```

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

Listing 25: /etc/faucet/facuet.yaml

```yaml
routerid: '10.0.1.3'  # Faucet's Unique ID.
server_addresses: ['10.0.1.3']  # Faucet's listen IP for BGP
neighbor_addresses: ['10.0.1.2']  # Neighbouring IP addresses (IPv4/IPv6)
neighbor_as: 65001  # Neighbour's AS number
```

Now reload Faucet.

`sudo systemctl reload faucet`

We can use the command line tool `birdc` to query the status of our peering connection, we should see that it is now established:

```
as_ns bgp birdc show protocols all faucet
```

<table>
<thead>
<tr>
<th>name</th>
<th>proto</th>
<th>table</th>
<th>state</th>
<th>since</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet</td>
<td>BGP</td>
<td>master</td>
<td>up</td>
<td>13:25:38</td>
<td>Established</td>
</tr>
</tbody>
</table>

Preference: 100
Input filter: ACCEPT
Output filter: ACCEPT
Routes: 1 imported, 1 exported, 1 preferred
Route change stats: received rejected filtered ignored accepted
Import updates: 1 0 0 0 1
Import withdraws: 0 0 --- 0 0
Export updates: 2 1 0 --- 1
Export withdraws: 0 --- --- --- 0

BGP state: Established
Neighbor address: 10.0.1.3
Neighbor AS: 65000
Neighbor ID: 10.0.1.3
Neighbor caps: AS4
Session: external AS4
Source address: 10.0.1.2
Hold timer: 185/240
Keepalive timer: 57/80

Using `birdc` we can also check what routes are being exported to faucet:
as ns bgp birdc show route export faucet

192.0.2.0/24 via 10.0.1.1 on veth0 [static1 13:25:34] * (200)

And which routes bird receives from faucet:

as ns bgp birdc show route protocol faucet

10.0.1.0/24 via 10.0.1.254 on veth0 [faucet 13:25:38 from 10.0.1.3] * (100) [i]

In /var/log/faucet/faucet.log we should now see log messages relating to BGP:

Listing 26: /var/log/faucet/faucet.log

Jan 16 13:25:17 faucet INFO Reloading configuration
Jan 16 13:25:17 faucet INFO configuration /etc/faucet/faucet.yaml changed, →analyzing differences
Jan 16 13:25:17 faucet INFO Add new datapath DPID 1 (0x1)
Jan 16 13:25:17 faucet INFO Adding BGP speaker key DP ID: 1, VLAN VID: 200, →IP version: 4 for VLAN servers vid:200 untagged: Port 3,Port 4,Port 5
Jan 16 13:25:38 faucet.INFO BGP peer router ID 10.0.1.2 AS 65001 up
Jan 16 13:25:38 faucet.INFO BGP add 192.0.2.0/24 nexthop 10.0.1.1
Jan 16 13:25:42 faucet.valve.INFO DPID 1 (0x1) br0 resolving 10.0.1.1 (1 flows) →on VLAN 200
Jan 16 13:25:42 faucet.valve.INFO DPID 1 (0x1) br0 adding new route 192.0.2.0/24 →via 10.0.1.1 (aa:97:cd:33:74:a9) on VLAN 200

Once confirming the BGP connection is up between BIRD and faucet and the correct routes are being advertised, we should now be able to ping the IP alias on the server namespace again:

as ns host1 ping 192.0.2.1

1.2.5 Stacking tutorial

Faucet has two primary modes of operation: independent switching and distributed switching.

In independent mode each decision about the network (learning, routing, etc) is made in the context of each individual switch.

This tutorial will cover Faucet’s distributed switching (a.k.a stacking) mode. Stacking allows decisions such as switching and routing to be made in the context of the whole network. This has great benefits for building resilient network topologies that can automatically recover from switch and port/cable failures. In this tutorial we will cover some of the new features and demonstrate how they work.

Prerequisites

- Knowledge of the VLAN and routing tutorial topics (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 network namespace
create_ns () {
    NAME=$1
    IP=$2
    NETNS=faucet-$(echo $NAME)
    sudo ip netns add $NETNS
    sudo ip link add dev veth-$(echo $NAME) type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-$(echo $NAME) up
    as_ns $NAME ip link set dev lo up
    [ -n "$IP" ] && as_ns $NAME ip addr add dev veth0 $IP
    as_ns $NAME ip link set dev veth0 up
}

# Run command inside network namespace
as_ns () {
    NAME=$1
    NETNS=faucet-$(echo $NAME)
    shift
    sudo ip netns exec $NETNS $@
}

# Add inter-switch link between two switches
inter_switch_link () {
    SW_A_NAME=$(echo $1 | cut -d ':' -f 1)
    SW_A_PORT=$(echo $1 | cut -d ':' -f 2)
    SW_B_NAME=$(echo $2 | cut -d ':' -f 1)
    SW_B_PORT=$(echo $2 | cut -d ':' -f 2)
    VETH_A=l-$(echo $SW_A_NAME)_$(echo $SW_A_PORT)-$(echo $SW_B_NAME)_$(echo $SW_B_PORT)
    VETH_B=l-$(echo $SW_B_NAME)_$(echo $SW_B_PORT)-$(echo $SW_A_NAME)_$(echo $SW_A_PORT)
    VETH_A=$(echo $VETH_A:0:15)
    VETH_B=$(echo $VETH_B:0:15)
    sudo ip link add dev $(echo $VETH_A) type veth peer name $(echo $VETH_B)
    sudo ip link set dev $(echo $VETH_A) up
    sudo ip link set dev $(echo $VETH_B) up
    sudo ovs-vsctl add-port $(echo $SW_A_NAME) $(echo $VETH_A) -- set interface $(echo $VETH_A) ofport_request=$(echo $SW_A_PORT)
    sudo ovs-vsctl add-port $(echo $SW_B_NAME) $(echo $VETH_B) -- set interface $(echo $VETH_B) ofport_request=$(echo $SW_B_PORT)
}

# Clean up namespaces, bridges and processes created during faucet tutorial
cleanup () {
    for NETNS in $(sudo ip netns list | grep "faucet-" | awk '{print $1}'); do
        if [ -n "$NETNS" ] || continue
            NAME=$(echo $NETNS|cut -d '-' -f 1)
            if [ -f "/run/dhclient-$NAME.pid" ]; then
                # Stop dhclient
                sudo pkill -F "/run/dhclient-$NAME.pid"
            fi
            if [ -f "/run/iperf3-$NAME.pid" ]; then
                # Stop iperf3
                sudo pkill -F "/run/iperf3-$NAME.pid"
            fi
    done
}

(continues on next page)
if [ -f "/run/bird-$NAME.pid" ]; then
    # Stop bird
    sudo pkill -F "/run/bird-$NAME.pid"
fi

# Remove netns and veth pair
sudo ip link delete veth-$NAME
sudo ip netns delete $NETNS

done

for isl in $(ip -o link show | awk -F': ' '{print $2}' | grep -oE "^l-br[0-9]-br[0-9]+.*$"); do
    # Delete inter-switch links
    sudo ip link delete dev $isl 2>/dev/null || true
done

for DNSMASQ in /run/dnsmasq-vlan*.pid; do
    [ -e "$DNSMASQ" ] || continue
    # Stop dnsmasq
    sudo pkill -F "$DNSMASQ"
done

# Remove faucet dataplane connection
sudo ip link delete veth-faucet 2>/dev/null || true

# Remove openvswitch bridges
sudo ovs-vsctl --if-exists del-br br0
sudo ovs-vsctl --if-exists del-br br1
sudo ovs-vsctl --if-exists del-br br2
sudo ovs-vsctl --if-exists del-br br3

• Run the cleanup script to remove old namespaces and switches:

    cleanup

Basic stacking

We can start by considering two switches with one host on each switch on the same VLAN.

Let’s define a simple base faucet.yaml to get started:
Now lets signal faucet to reload the configuration file.

```
sudo systemctl reload faucet
```

We need to create our two hosts, host1 and host2.

```
create_ns host1 10.0.1.1/24
create_ns host2 10.0.1.2/24
```

To setup multiple switches in Open vSwitch we can define two bridges with different datapath-ids and names. We’ll be using br0 and br1.

```
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 \
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br1 \
-- set bridge br1 other-config:datapath-id=0000000000000002 \
-- set bridge br1 other-config:disable-in-band=true \
-- set bridge br1 fail_mode=secure \
-- add-port br1 veth-host2 -- set interface veth-host2 ofport_request=1 \
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

Since the switches are not connected it will be impossible to ping between the two hosts.

```
as_ns host1 ping 10.0.1.2
```

To connect the switches we can use the Faucet switch stacking feature. First, we need to define a root switch for our stack by setting a stack priority value for br0, the datapath with the lowest priority will be root. Second, we need to add stack interfaces connecting each datapath, we do this by defining the stack parameter on an interface. When defining a stack interface we say which datapath (dp) and port the other end of the cable is connected to.

Replace your base faucet.yaml from earlier with this version with stacking enabled:

```
vlans:
  hosts:
    vid: 100
dps:
  br0:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        description: "host1 network namespace"
        native_vlan: hosts
  br1:
    dp_id: 0x2
    hardware: "Open vSwitch"
    interfaces:
      1:
        description: "host2 network namespace"
        native_vlan: hosts
```
To connect two Open vSwitch bridges we can use a veth interface pair. We will use the `inter_switch_link` function we defined earlier to connect br0 port 2 to br1 port 2:

```bash
inter_switch_link br0:2 br1:2
```

Let’s reload Faucet and see what happens.

```bash
sudo systemctl reload faucet
```

Faucet will start sending out LLDP beacons to connect up the stack ports. We can see this happening in the log file when the switches report that port 2 (the stack port) is UP.

```
DPID 2 (0x2) br1 LLDP on 0e:00:00:00:00:00:01, Port 2 from 0e:00:00:00:00:00:01 (remote_
  →DPID 1 (0x1), port 2) state 2
DPID 2 (0x2) br1 Stack Port 2 INIT
DPID 1 (0x1) br0 LLDP on 0e:00:00:00:00:00:01, Port 2 from 0e:00:00:00:00:00:01 (remote_
  →DPID 2 (0x2), port 2) state 2
DPID 1 (0x1) br0 Stack Port 2 INIT
DPID 2 (0x2) br1 LLDP on 0e:00:00:00:00:00:01, Port 2 from 0e:00:00:00:00:00:01 (remote_
  →DPID 1 (0x1), port 2) state 1
DPID 2 (0x2) br1 Stack Port 2 UP
DPID 2 (0x2) br1 1 stack ports changed state
```
Note: If we were to accidentally cable our switches incorrectly faucet would report the incorrect cabling in the log file.

Now that the two switches are connected and our stack is up, we can ping between the two hosts.

```bash
as_ns host1 ping 10.0.1.2
```

### Inter-VLAN routing with stacking

For this task we will see that inter-VLAN routing can work between hosts on different switches.

First run the cleanup.

```bash
cleanup
```

We can accomplish inter-VLAN routing between different switches by using the stacking feature. To do this we will be combining the methods from the Basic stacking and the Routing between VLANs tutorials. However, we need to set drop_spoofed_faucet_mac to false on each DP. Doing this will prevent a packet that has been routed and come from a stack port from being dropped.

Here is a full faucet.yaml you can copy and paste that sets up our stack topology and enables all the features we need.

```yaml
vlans:
  hosts:
    vid: 100
    faucet_vips: ["10.0.1.254/24"]
    faucet_mac: "00:00:00:00:00:11"
  servers:
    vid: 200
    faucet_vips: ["10.0.2.254/24"]
```

(continues on next page)
```python
faucet_mac: "00:00:00:00:00:22"

routers:
  router-1:
    vlans: [hosts, servers]

dps:
  br0:
    dp_id: 0x1
    hardware: "Open vSwitch"
    stack:
      priority: 1
    drop_spoofed_faucet_mac: False
    interfaces:
      1:
        description: "host1 network namespace"
        native_vlan: hosts
      2:
        description: "br0 stack link to br1"
        stack:
          dp: br1
          port: 2
      3:
        description: "server1 network namespace"
        native_vlan: servers

br1:
  dp_id: 0x2
  hardware: "Open vSwitch"
  drop_spoofed_faucet_mac: False
  interfaces:
    1:
      description: "host2 network namespace"
      native_vlan: hosts
    2:
      description: "br1 stack link to br0"
      stack:
        dp: br0
        port: 2
    3:
      description: "server2 network namespace"
      native_vlan: servers
```

Reload faucet to enable inter-VLAN routing.

```
sudo systemctl reload faucet
```

As we have learnt previously. First, set up the hosts:

```
create_ns host1 10.0.1.1/24
create_ns host2 10.0.1.2/24
create_ns server1 10.0.2.1/24
create_ns server2 10.0.2.2/24
```

Now we can set-up the default routes for each host.

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

(continues on next page)
Next, we can create the bridges.

```bash
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-server1 -- set interface veth-server1 ofport_request=3 \ 
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

```bash
sudo ovs-vsctl add-br br1 \
-- set bridge br1 other-config:datapath-id=0000000000000002 \ 
-- set bridge br1 other-config:disable-in-band=true \ 
-- set bridge br1 fail_mode=secure \ 
-- add-port br1 veth-host2 -- set interface veth-host2 ofport_request=1 \ 
-- add-port br1 veth-server2 -- set interface veth-server2 ofport_request=3 \ 
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

And finally, we can create the inter-switch links to connect the bridges to each other.

```
inter_switch_link br0:2 br1:2
```

Now it should be possible to ping between any combination of hosts on any VLAN after the LLDP has configured the stack ports as UP. For example host1 can ping to server1 on the same switch as well as server2 on the other switch via the use of the stack link.

```
as_ns host1 ping 10.0.2.1
as_ns host1 ping 10.0.2.2
```

**Tunneling over a stack**

Faucet has a feature that allows us to tunnel packets from one datapath to another without having to think about the underlying network topology. In this example we have three switches and two hosts. We will create a tunnel that runs over top of this topology connecting host1 and host2 together.

First run the cleanup.

```
cleanup
```

Now let’s define our faucet.yaml that will make this network work. The configuration file below defines our faucet stack topology and ports for our host1 and host2. An important thing to note is that we define our two hosts on separate VLANs so they should not be able to communicate.
The other thing to notice is the two ACLs we define, `tunnel-to-host1` and `tunnel-to-host2`. At the moment these ACLs match all traffic (though we could easily add a match here to only tunnel a subset of traffic, see **ACL tutorial** for more details). Each tunnel sets the destination datapath and port for traffic matching the ACL, we currently support one type of tunnel, VLAN, and must reserve a tunnel VLAN here using the `tunnel_id` parameter (in future we could support different types of tunnels).

The two ACLs are then applied to the ports host1 and host2 are connected to.

```
Listing 31: /etc/faucet/faucet.yaml

```acl:`

```yaml
acls:
  tunnel-to-host1:
    - rule:
      actions:
        output:
          tunnel:
            type: 'vlan'
            tunnel_id: 901
            dp: br0
            port: 1
  tunnel-to-host2:
    - rule:
      actions:
        output:
          tunnel:
            type: 'vlan'
            tunnel_id: 902
            dp: br2
            port: 1

vlans:
  host1:
    vid: 101
  host2:
    vid: 102

dps:
  br0:
    dp_id: 0x1
    hardware: "Open vSwitch"
    stack:
      priority: 1
    interfaces:
      1:
        description: "host1 network namespace"
        native_vlan: host1
        acl_in: tunnel-to-host2
  2:
    description: "br0 stack link to br1"
    stack:
      dp: br1
      port: 1

  br1:
    dp_id: 0x2
    hardware: "Open vSwitch"
    interfaces:
      1:
        description: "br1 stack link to br0"
        stack:
          dp: br0
```

(continues on next page)
When we have updated our configuration to match above, signal to faucet to reload the configuration file.

```
sudo systemctl reload faucet
```

Then we can set up the hosts:

```
create_ns host1 10.0.1.1/24
create_ns host2 10.0.1.2/24
```

Next, we can create the bridges.

```
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 \  
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br1 \  
-- set bridge br1 other-config:datapath-id=0000000000000002 \  
-- set bridge br1 other-config:disable-in-band=true \  
-- set bridge br1 fail_mode=secure \  
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br2 \  
-- set bridge br2 other-config:datapath-id=0000000000000003 \  
-- set bridge br2 other-config:disable-in-band=true \  
-- set bridge br2 fail_mode=secure \  
-- add-port br2 veth-host2 -- set interface veth-host2 ofport_request=1 \  
-- set-controller br2 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

We also need to define inter-switch links that connect br0 and b1 as well as br1 and br2.

```
inter_switch_link br0:2 br1:1
inter_switch_link br1:2 br2:2
```

We should now be able to ping between host1 and host2 despite them being on different VLANs and datapaths because
of the tunnel.

```
as_ns host1 ping 10.0.1.2
```

The reason the hosts can now communicate is that faucet is using the stack topology to find a path between the two hosts and automatically stitching up a tunnel. If we had a more complicated topology with multiple valid paths between the hosts, faucet will pick one and if the topology changes faucet will ensure the tunnel still goes over a valid path.

If we were to disable the ACLs on the port we would notice the hosts would no longer be able to ping.

### Redundant stack links

Faucet is able to handle stack topologies with loops in them. This is because when faucet brings up a stack topology for the first time (or when it detects the network topology has changed), it has enough knowledge of the network to calculate a spanning tree for the network without the need for running a spanning tree protocol. Faucet uses this spanning tree to ensure broadcast packets aren’t looped around the network.

This feature enables us to build fault-tolerant network architectures that can survive switch/port failures, a simple example is a ring topology:

![Ring Topology Diagram](image-url)

To build this network, let’s first cleanup from previous exercises.

```
cleanup
```

We should be quite familiar with configuring faucet for stacks now, let’s define a faucet.yaml that matches our ring topology.

```
Listing 32: /etc/faucet/faucet.yaml

vlans:
  hosts:
    vid: 100
dps:
  br0:
    dp_id: 0x1
```

(continues on next page)
Reload faucet to enable the ring topology.

```
sudo systemctl reload faucet
```

We will define three hosts, one on each switch.
create_ns host1 10.0.1.1/24
create_ns host2 10.0.1.2/24
create_ns host3 10.0.1.3/24

Now let’s define the three switches.

```
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 \ 
  -- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br1 \
  -- set bridge br1 other-config:datapath-id=0000000000000002 \ 
  -- set bridge br1 other-config:disable-in-band=true \ 
  -- set bridge br1 fail_mode=secure \ 
  -- add-port br1 veth-host2 -- set interface veth-host2 ofport_request=1 \ 
  -- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br2 \
  -- set bridge br2 other-config:datapath-id=0000000000000003 \ 
  -- set bridge br2 other-config:disable-in-band=true \ 
  -- set bridge br2 fail_mode=secure \ 
  -- add-port br2 veth-host3 -- set interface veth-host3 ofport_request=1 \ 
  -- set-controller br2 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```

We also need to create the inter-switch links forming our ring network.

```
inter_switch_link br0:2 br1:2
inter_switch_link br0:3 br2:2
inter_switch_link br1:3 br2:3
```

Once the network is up we should be able to ping from all hosts to all other hosts.

```
as_ns host1 ping 10.0.1.2
as_ns host1 ping 10.0.1.3
```

Now let us intentionally introduce a fault into the network, our network should be able to survive a single cable failure and still have all devices reachable.

To test this we will manually disable the link between br0 and br2.

```
sudo ip link set down l-br0_3-br2_2
sudo ip link set down l-br2_2-br0_3
```

Which will force traffic between br0 and br2 to now go via br1, we can test this by ensuring host1 can still ping host3.

```
as_ns host1 ping 10.0.1.3
```
Multi-root stack

The previous exercise introduced the ability to survive cable failures, but you might have noticed in each exercise so far we have defined only a single root switch. If we were to lose this root switch the network would no longer function.

In this exercise we will introduce multi-root stacked networks which give us the ability to tolerate switch failures.

This example topology will allow us to survive any single cable failure or either of br0 or br1 failing.

Before we begin, let’s do another cleanup.

cleanup

Our faucet.yaml will look familiar here, except for one difference, we now have two switches defined as stack priority 1. This signals to faucet that it has two equal priority root candidates it can use when selecting a root for the network.

Listing 33: /etc/faucet/faucet.yaml

```yaml
vlans:
    hosts:
        vid: 100
dps:
    br0:
        dp_id: 0x1
        hardware: "Open vSwitch"
        stack:
            priority: 1
        interfaces:
            1:
                description: "br0 stack link to br2"
                stack:
                    dp: br2
                    port: 2
```

(continues on next page)
When we have this new faucet.yaml loaded we will do a full restart this time instead of reloading to force a root
Faucet Documentation

election.

```bash
sudo systemctl restart faucet
```

We will create some hosts to let us test the failure scenarios of this topology.

```bash
create_ns host1 10.0.1.1/24
create_ns host2 10.0.1.2/24
```

We also need to define our four switches.

```bash
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 \
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br1 \
-- set bridge br1 other-config:datapath-id=0000000000000002 \
-- set bridge br1 other-config:disable-in-band=true \
-- set bridge br1 fail_mode=secure \
-- set-controller br1 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br2 \
-- set bridge br2 other-config:datapath-id=0000000000000003 \
-- set bridge br2 other-config:disable-in-band=true \
-- set bridge br2 fail_mode=secure \
-- add-port br2 veth-host1 -- set interface veth-host1 ofport_request=1 \
-- set-controller br2 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

sudo ovs-vsctl add-br br3 \
-- set bridge br3 other-config:datapath-id=0000000000000004 \
-- set bridge br3 other-config:disable-in-band=true \
-- set bridge br3 fail_mode=secure \
-- add-port br3 veth-host2 -- set interface veth-host2 ofport_request=1 \
-- set-controller br3 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654

We need to fully mesh br0, br1, br2 and br3 to match our topology diagram above.

```bash
# Inter-switch links for br0
inter_switch_link br0:1 br2:2
inter_switch_link br0:2 br3:3

# Inter-switch links for br1
inter_switch_link br1:1 br3:2
inter_switch_link br1:2 br2:3
```

When everything is setup we should be able to ping between host1 and host2.

```bash
as_ns host1 ping 10.0.1.2
```

Now let’s inspect the log file to find out which switch is currently our root.

```bash
$ grep -ai "stack root changed" /var/log/faucet/faucet.log | tail -n 1
Oct 08 04:19:24 faucet INFO stack root changed from None to br0
```

Since br0 is the switch which is currently root, let’s delete it to simulate a switch failure.
sudo ovs-vsctl del-br br0

If we look into the log file we should see faucet detects the switch is down and br1 takes over as the new root.

Listing 34: /var/log/faucet/faucet.yaml

<table>
<thead>
<tr>
<th>Time</th>
<th>Log Level</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 08 04:22:52</td>
<td>WARNING</td>
<td>DPID 1 (0x1) br0 datapath down</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 1 (0x1) br0 LLDP for Port 1 inactive after ~17s</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 1 (0x1) br0 LLDP for Port 2 inactive after ~17s</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>ERROR</td>
<td>DPID 1 (0x1) br0 Stack Port 1 DOWN, too many ~3 packets lost, last received 17s ago</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 2 (0x2) br1 shortest path to root is via Port 1</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 3 (0x3) br2 shortest path to root is via Port 2</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 4 (0x4) br3 shortest path to root is via Port 3</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>ERROR</td>
<td>DPID 1 (0x1) br0 Stack Port 2 DOWN, too many ~3 packets lost, last received 17s ago</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 2 (0x2) br1 shortest path to root is via Port 1</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 3 (0x3) br2 shortest path to root is via Port 2</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 4 (0x4) br3 shortest path to root is via Port 3</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>ERROR</td>
<td>DPID 1 (0x1) br0 Stack Port 3 DOWN, too many ~3 packets lost, last received 17s ago</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 2 (0x2) br1 shortest path to root is via Port 1</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 3 (0x3) br2 shortest path to root is via Port 2</td>
</tr>
<tr>
<td>Oct 08 04:23:03</td>
<td>INFO</td>
<td>DPID 4 (0x4) br3 shortest path to root is via Port 3</td>
</tr>
<tr>
<td>Oct 08 04:23:15</td>
<td>INFO</td>
<td>stack root changed from br0 to br1</td>
</tr>
<tr>
<td>Oct 08 04:23:15</td>
<td>INFO</td>
<td>root now br1 (all candidates ('br0', 'br1'), healthy [~'br1'])</td>
</tr>
</tbody>
</table>

We should also still be able to ping between host1 and host2 after the stack has recalculated.

as_ns host1 ping 10.0.1.2
1.2.6 NFV services tutorial

This tutorial will cover using faucet with Network Function Virtualisation (NFV) style services. NFV services that will be demonstrated in this tutorial are:

- DHCP/DNS server
- Zeek (formerly known as Bro) Intrusion Detection System (IDS)

This tutorial demonstrates how the previous topics in this tutorial series can be combined to run real world 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’.

```bash
# Run command inside network namespace
as_ns () {
    NAME=$1
    NETNS=faucet-$NAME
    shift
    sudo ip netns exec $NETNS @
}

# Create network namespace
create_ns () {
    NAME=$1
    IP=$2
    NETNS=faucet-$NAME
    sudo ip netns add $NETNS
    sudo ip link add dev veth-$NAME type veth peer name veth0 netns $NETNS
    sudo ip link set dev veth-$NAME up
    as_ns $NAME ip link set dev lo up
    [ -n "$IP" ] && as_ns $NAME ip addr add dev veth0 $IP
    as_ns $NAME ip link set dev veth0 up
}

# Clean up namespaces, bridges and processes created during faucet tutorial
cleanup () {
    for NETNS in $(sudo ip netns list | grep "faucet-" | awk '{print $1}'); do
        [ -n "$NETNS" ] || continue
        NAME=${NETNS#faucet-}
        if [ -f "/run/dhclient-$NAME.pid" ]; then
            # Stop dhclient
            sudo pkill -F "/run/dhclient-$NAME.pid"
        fi
        if [ -f "/run/iperf3-$NAME.pid" ]; then
            fi
    done
}
```

(continues on next page)
# Stop iperf3
    sudo pkill -F "/run/iperf3-$\{NAME\}.pid"
fi
if [ -f "/run/bird-$\{NAME\}.pid" ]; then
    # Stop bird
    sudo pkill -F "/run/bird-$\{NAME\}.pid"
fi
    # Remove netns and veth pair
    sudo ip link delete veth-$\{NAME\}
    sudo ip netns delete $\{NETNS\}
done
for isl in 
    $(ip -o link show | awk -F': ' '{print $2}' | grep -oE "^l-\<-br[0-9]?\(...\)\-br[0-9]?\(...\)?$$"); do
    # Delete inter-switch links
    sudo ip link delete dev $isl 2>/dev/null || true
done
for DNSMASQ in /run/dnsmasq-vlan*.pid; do
    [ -e "$\{DNSMASQ\}" ] || continue
    # Stop dnsmasq
    sudo pkill -F "$\{DNSMASQ\}"
done
    # Remove faucet dataplane connection
    sudo ip link delete veth-faucet 2>/dev/null || true
    # Remove openvswitch bridges
    sudo ovs-vsctl --if-exists del-br br0
    sudo ovs-vsctl --if-exists del-br br1
    sudo ovs-vsctl --if-exists del-br br2
    sudo ovs-vsctl --if-exists del-br br3
\}

# Add tagged VLAN interface to network namespace
add_tagged_interface () { 
    NAME=$1
    VLAN=$2
    IP=$3
    NETNS=faucet-$\{NAME\}
    as_ns $\{NAME\} ip link add link veth0 name veth0.$\{VLAN\} type vlan id $\{VLAN\}
    [ -n "$\{IP\}" ] && as_ns $\{NAME\} ip addr add dev veth0.$\{VLAN\} $\{IP\}
    as_ns $\{NAME\} ip link set dev veth0.$\{VLAN\} up
    as_ns $\{NAME\} ip addr flush dev veth0
\}

• Run the cleanup script to remove old namespaces and switches:

    cleanup
Network setup

The network will be divided into three VLANs, two of which are client VLANs (200 & 300), with two clients in each and a DHCP/DNS server. There is also a separate VLAN 100 for the Zeek server which we will mirror traffic two from the other two VLANs.

To start, let's create our hosts and dnsmasq namespaces.

```
# DHCP/DNS server
create_ns dnsmasq 0.0.0.0
add_tagged_interface dnsmasq 200 192.168.2.1/24 # to serve VLAN 200
add_tagged_interface dnsmasq 300 192.168.3.1/24 # to serve VLAN 300

# VLAN 200 hosts
create_ns host1 0.0.0.0
create_ns host2 0.0.0.0
# VLAN 300 hosts
create_ns host3 0.0.0.0
create_ns host4 0.0.0.0
```

Then create an Open vSwitch bridge 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-dnsmasq -- set interface veth-dnsmasq ofport_request=5 
-- set-controller br0 tcp:127.0.0.1:6653 tcp:127.0.0.1:6654
```
Dnsmasq setup

We will use dnsmasq to assign IP addresses to our host namespaces via the DHCP protocol. It will also serve as our DNS resolver for the hosts.

First install dnsmasq:

```bash
sudo apt-get install dnsmasq
sudo systemctl stop dnsmasq
```

Run the following two commands to start two dnsmasq instances inside the dnsmasq namespace. One instance will serve hosts on VLAN 200 and the other VLAN 300. We will be providing DHCP leases in the supplied ranges, the lease will set the gateway for each host to point at faucet’s virtual IP and set dnsmasq as the DNS resolver. We also provide a fake `does.it.work` DNS name which we will later use to demonstrate DNS is working as expected.

```bash
# 192.168.2.0/24 for VLAN 200
as_ns dnsmasq dnsmasq \
   --dhcp-range=192.168.2.10,192.168.2.20 \  
   --dhcp-sequential-ip \  
   --dhcp-option=option:router,192.168.2.254 \  
   --no-resolv \  
   --txt-record=does.it.work,yes \  
   --bind-interfaces \  
   --except-interface=lo --interface=veth0.200 \  
   --dhcp-leasefile=/tmp/nfv-dhcp-vlan200.leases \  
   --log-facility=/tmp/nfv-dhcp-vlan200.log \  
   --pid-file=/run/dnsmasq-vlan200.pid \  
   --conf-file=  
# 192.168.3.0/24 for VLAN 300
as_ns dnsmasq dnsmasq \
   --dhcp-range=192.168.3.10,192.168.3.20 \  
   --dhcp-sequential-ip \  
   --dhcp-option=option:router,192.168.3.254 \  
   --no-resolv \  
   --txt-record=does.it.work,yes \  
   --bind-interfaces \  
   --except-interface=lo --interface=veth0.300 \  
   --dhcp-leasefile=/tmp/nfv-dhcp-vlan300.leases \  
   --log-facility=/tmp/nfv-dhcp-vlan300.log \  
   --pid-file=/run/dnsmasq-vlan300.pid \  
   --conf-file=
```

Now let’s configure faucet.yaml.

Listing 35: `/etc/faucet/faucet.yaml`

```yaml
vlans:
  vlan200:
    vid: 200
    description: "192.168.2.0/24 network"
    faucet_vips: ["192.168.2.254/24"]
    faucet_mac: "00:00:00:00:00:22"
  vlan300:
    vid: 300
    description: "192.168.3.0/24 network"
    faucet_vips: ["192.168.3.254/24"]
    faucet_mac: "00:00:00:00:00:33"
```

(continues on next page)
dps:
    sw1:
        dp_id: 0x1
        hardware: "Open vSwitch"
        interfaces:
            1:
                name: "host1"
                description: "host1 network namespace"
                native_vlan: vlan200
            2:
                name: "host2"
                description: "host2 network namespace"
                native_vlan: vlan200
            3:
                name: "host3"
                description: "host3 network namespace"
                native_vlan: vlan300
            4:
                name: "host4"
                description: "host4 network namespace"
                native_vlan: vlan300
            5:
                name: "dnsmasq"
                description: "dnsmasq server network namespace"
                tagged_vlans: [vlan200, vlan300]

Now reload faucet configuration file.

```
sudo systemctl reload faucet
```

Use dhclient to configure host1 to host4 using DHCP (it may take a few seconds, but should return when successful).

```
as_ns host1 dhclient -v -pf /run/dhclient-host1.pid -lf /run/dhclient-host1.leases -veth0
as_ns host2 dhclient -v -pf /run/dhclient-host2.pid -lf /run/dhclient-host2.leases -veth0
as_ns host3 dhclient -v -pf /run/dhclient-host3.pid -lf /run/dhclient-host3.leases -veth0
as_ns host4 dhclient -v -pf /run/dhclient-host4.pid -lf /run/dhclient-host4.leases -veth0
```

If dhclient is unable to obtain an address you can check `/tmp/nfv-dhcp-vlan<vlanid>.log` (e.g. `/tmp/nfv-dhcp-vlan300.leases`) to check the log messages from dnsmasq.

To look up the address for each namespace we can run the following commands:

```
as_ns host1 ip address show dev veth0
as_ns host2 ip address show dev veth0
as_ns host3 ip address show dev veth0
as_ns host4 ip address show dev veth0
```

If the hosts have IPs then great our DHCP server works.

At the moment we should be able to ping inside VLAN 200 and VLAN 300:

```
as_ns host1 ping <ip of host2> # both in VLAN 200 should work
as_ns host3 ping <ip of host4> # both in VLAN 300 should work
```
Pinging between VLANs will not currently work as we didn’t turn on inter-VLAN routing in our faucet configuration.

**DNS**

We can use faucet to enforce where protocols such as DNS go on the network. In this section we will use a faucet ACL to rewrite DNS packets to allow our dnsmasq namespace to answer DNS queries for any IP address.

Firstly, we can see that our dnsmasq server is correctly responding to DNS requests by manually querying them:

```
as_ns host1 host -t txt does.it.work 192.168.2.1
as_ns host3 host -t txt does.it.work 192.168.3.1
```

Both commands should return:

```
does.it.work descriptive text "yes"
```

But if we tried to query say 8.8.8.8 we would see this fail:

```
as_ns host1 host -t txt does.it.work 8.8.8.8
```

To make this work we first need the MAC address of the dnsmasq container:

```
as_ds dnsmasq cat /sys/class/net/veth0/address
00:11:22:33:44:55
```

We now replace our previous faucet configuration with the configuration below which adds an ACL that rewrites the MAC address of all DNS packets from the host namespaces and sends these to our dnsmasq namespace. Make sure to update the example MAC address of 00:11:22:33:44:55 with the one you get from running the previous command.

```
Listing 36: /etc/faucet/faucet.yaml
```

```yaml
vlans:
  vlan200:
    vid: 200
    description: "192.168.2.0/24 network"
    faucet_vips: ["192.168.2.254/24"]
    faucet_mac: "00:00:00:00:00:22"
  vlan300:
    vid: 300
    description: "192.168.3.0/24 network"
    faucet_vips: ["192.168.3.254/24"]
    faucet_mac: "00:00:00:00:00:33"
dps:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: vlan200
        acls_in: [nfv-dns, allow-all]
      2:
        name: "host2"
        description: "host2 network namespace"
```

(continues on next page)
native_vlan: vlan200
acls_in: [nfv-dns, allow-all]

3:
  name: "host3"
description: "host3 network namespace"
native_vlan: vlan300
acls_in: [nfv-dns, allow-all]

4:
  name: "host4"
description: "host4 network namespace"
native_vlan: vlan300
acls_in: [nfv-dns, allow-all]

5:
  name: "dnsmasq"
description: "dnsmasq server network namespace"
tagged_vlans: [vlan200, vlan300]

acls:
  nfv-dns:
    # Force UDP DNS to our DNS server
    - rule:
      dl_type: 0x800  # ipv4
      nw_proto: 17    # udp
      udp_dst: 53     # dns
      actions:
        output:
          set_fields:
            - eth_dst: "00:11:22:33:44:55"  # MAC address of dnsmasq
        namespace
        allow: True
        # Force TCP DNS to our DNS server
        - rule:
          dl_type: 0x800  # ipv4
          nw_proto: 6     # tcp
          tcp_dst: 53     # dns
          actions:
            output:
              set_fields:
                - eth_dst: "00:11:22:33:44:55"  # MAC address of dnsmasq
            namespace
            allow: True
  allow-all:
    - rule:
      actions:
        allow: True

As usual reload faucet configuration file.

```
sudo systemctl reload faucet
```

The next step is to configure the namespace to be able to handle incoming DNS packets with any IP, this can be done by adding some rules to iptables that will NAT all DNS traffic to the IP address of the VLAN interface:

```
as_ns dnsmasq iptables -t nat -A PREROUTING -i veth0.200 -p udp --dport 53 -j DNAT --to-destination 192.168.2.1
as_ns dnsmasq iptables -t nat -A PREROUTING -i veth0.200 -p tcp --dport 53 -j DNAT --to-destination 192.168.2.1
```
(continues on next page)
Now we should be able to query any IP address from the hosts and get a valid DNS response:

```
as_ns host1 host -t txt does.it.work 8.8.8.8
as_ns host2 host -t txt does.it.work 8.8.4.4
```

**Zeek IDS**

We will now add an IDS to our network on it's on separate VLAN and use faucet to mirror packets from VLAN 200 and 300 to the IDS VLAN.

**Zeek installation**

We need first to install Zeek (formerly known as Bro).

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

**Configure Zeek**

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

```
Listing 37: /etc/bro/node.cfg

[bro]
type=standalone
host=localhost
interface=veth0
```

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

```
Listing 38: /etc/bro/broctl.cfg

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

**Run Zeek**

Firstly, let’s create a namespace to run Zeek inside:

```
create_ns zeek 192.168.0.1
sudo ovs-vsctl add-port br0 veth-zeek -- set interface veth-zeek ofport_request=6
```

Since this is the first-time use of the Zeek command shell application, perform an initial installation of the BroControl configuration:
as_ns zeek broctl install

Then start Zeek instant

as_ns zeek broctl start

Check Zeek status

as_ns zeek broctl status

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Host</th>
<th>Status</th>
<th>Pid</th>
<th>Started</th>
</tr>
</thead>
<tbody>
<tr>
<td>bro</td>
<td>standalone</td>
<td>localhost</td>
<td>running</td>
<td>15052</td>
<td>07 May 09:03:59</td>
</tr>
</tbody>
</table>

Now let’s add a mirror ACL so all VLAN 200 & VLAN 300 traffic is sent to Zeek.

We will use a VLAN ACLs similar to the previous VLAN tutorial. Copy and paste the entire configuration below into faucet.yaml.

Listing 39: /etc/faucet/faucet.yaml

```yaml
acls:
  mirror-acl:
    - rule:
        actions:
          allow: true
          mirror: zeek
vlans:
  zeek-vlan:
    vid: 100
    description: "Zeek IDS network"
  vlan200:
    vid: 200
    description: "192.168.2.0/24 network"
    faucet_vips: ["192.168.2.254/24"]
    faucet_mac: "00:00:00:00:00:22"
    acls_in: [mirror-acl]
  vlan300:
    vid: 300
    description: "192.168.3.0/24 network"
    faucet_vips: ["192.168.3.254/24"]
    faucet_mac: "00:00:00:00:00:33"
    acls_in: [mirror-acl]
dacls:
  sw1:
    dp_id: 0x1
    hardware: "Open vSwitch"
    interfaces:
      1:
        name: "host1"
        description: "host1 network namespace"
        native_vlan: vlan200
      2:
        name: "host2"
        description: "host2 network namespace"
        native_vlan: vlan200
      3:
        name: "host3"
        description: "host3 network namespace"
```

(continues on next page)
native_vlan: vlan300
4:
name: "host4"
description: "host4 network namespace"
native_vlan: vlan300
5:
name: "dnsmasq"
description: "dnsmasq server network namespace"
tagged_vlans: [vlan200, vlan300]
6:
name: "zeek"
description: "Zeek network namespace"
native_vlan: zeek-vlan

As usual reload faucet configuration file.

```bash
sudo systemctl reload faucet
```

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

```bash
as_ns host4 host -t txt does.it.work 192.168.3.1
```

Then if we inspect the Zeek logs for DNS `/var/log/bro/current/dns.log`, we should see that Zeek has seen the DNS queries and logged these.

Listing 40: `/var/log/bro/current/dns.log`

```plaintext
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path dns
#open 2019-01-17-17-43-56
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p
˓→ proto trans_id rtt query qclass qclass_name qtype qtype_name
˓→ name rcode rcode_name AA TC RD RA Z answers
˓→ TTLs rejected
#types time string addr port addr port enum count interval
˓→ string count string count string count string bool bool
˓→ bool count vector[string] vector[interval] bool
1547700236.794299 CsulWM1Px7fiyPpiCVi 192.168.3.10 43428 192.168.3.1
˓→ 53 udp 14288 0.006973does.it.work 1 C_INTERNET 16 TXT
˓→ 0 NOERROR T F T T 2 TXT 3 yes 0.
˓→ 000000 F
1547700379.311319 CZalioBd3CgWmgS8 192.168.3.1 45089 192.168.3.1
˓→ 53 udp 64001 0.00336does.it.work 1 C_INTERNET 16 TXT
˓→ 0 NOERROR T F T T 0 TXT 3 yes 0.
˓→ 000000 F
```

You can also check if the traffic is being mirrored as expected using `tcpdump` in the `zeek` network namespace:

```bash
as_ns zeek sudo tcpdump -i veth0 -n -l
```

in one window, and then generating some more DNS traffic, eg:

```bash
as_ns host4 host -t txt does.it.work 192.168.3.1
```
then you should see something like:

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:19:24.625109 IP 192.168.3.1.53 > 192.168.3.13.38174: 64571* 1/0/0 TXT "yes" (46)
```

### 1.3 Installation

We recommend installing faucet with apt for first time users and provide a *Installing faucet for the first time* tutorial which walks you through all the required steps for setting up faucet and gauge for the first time.

Once installed, see *Configuration* for documentation on how to configure faucet. Also, see *Vendor-specific Documentation* for documentation on how to configure your switch.

More advanced methods of installing faucet are also available here:

1. **Installation using APT**
2. **Installation with Docker**
3. **Installation with Pip**
4. **Installing on Raspberry Pi**
5. **Installing with Virtual Machine image**

#### 1.3.1 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:

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

**Installation on Debian/Raspbian 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_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:

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

## 1.3.2 Installation with Docker

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

### Installing docker

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

### Configuring dockers

First, we need to create some configuration files on our host to mount inside the docker containers to configure faucet and gauge:

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

See the Configuration section for configuration options.

### Starting dockers

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:

```bash
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 \
  -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.

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

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.yaml file.

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

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

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

Name: Prometheus  
Type: Prometheus  
Url: http://prometheus:9090

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

### 1.3.3 Installation with Pip

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

#### Installing faucet

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:

```bash
faucet --verbose
```

To run Gauge manually:

```bash
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:

```bash
faucet --help
```

Starting faucet With systemd

Systemd can be used to start Faucet and Gauge at boot automatically:

```bash
$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 42: faucet.service

```ini
[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 43: gauge.service

```ini
[Unit]
Description="Gauge OpenFlow statistics controller"
```
1.3.4 Installing on Raspberry Pi

We provide a Raspberry Pi image running FAUCET which can be retrieved from the latest faucet release page on GitHub. Download the faucet_VERSION_raspbian-lite.zip file.

The image can then be copied onto an SD card following the same steps from the official Raspberry Pi installation guide.

Once you have booted up the Raspberry Pi and logged in using the default credentials you can follow through the Installing faucet for the first time tutorial starting from Configure prometheus to properly configure each component.

**Note:** It is strongly recommended to use a Raspberry Pi 3 or better.

1.3.5 Installing with 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).

**Downloading pre-built images**

Pre-built images are available on github, see the latest faucet release page on GitHub and download the faucet-amd64-VERSION.qcow2 file.

**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:

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>22</td>
</tr>
<tr>
<td>Faucet OpenFlow Channel</td>
<td>6653</td>
</tr>
<tr>
<td>Gauge OpenFlow Channel</td>
<td>6654</td>
</tr>
<tr>
<td>Grafana Web Interface</td>
<td>3000</td>
</tr>
<tr>
<td>Prometheus Web Interface</td>
<td>9090</td>
</tr>
</tbody>
</table>

Default Credentials

<table>
<thead>
<tr>
<th>Service</th>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM TTY Console</td>
<td>faucet</td>
<td>faucet</td>
</tr>
<tr>
<td>SSH</td>
<td>faucet</td>
<td>faucet</td>
</tr>
<tr>
<td>Grafana Web Interface</td>
<td>admin</td>
<td>admin</td>
</tr>
</tbody>
</table>

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: http://localhost:9090
```

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 Configuration

1.4.1 Faucet configuration

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

Listing 44: faucet.yaml

```yaml
include:
  - acls.yaml

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

  guest:
    vid: 200
    description: "guest network"
    faucet_mac: "0e:00:00:00:20:01"
    faucet_vips: ['10.0.200.254/24', '2001:200::1/64', 'fe80::c00:00ff:fe00:2001/64']

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

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

(continues on next page)
description: "services container"
native_vlan: office
acls_in: [service-port-protect]

5:
  name: "trunk"
description: "VLAN trunk to sw2"
tagged_vlans: [office]
acls_in: [service-port-protect]

sw2:
  dp_id: 0x2
  hardware: "Allied-Telesis"
  interfaces:
    1:
      name: "pi"
description: "raspberry pi"
native_vlan: office
acls_in: [access-port-protect]

    2:
      name: "laptop"
description: "guest laptop"
native_vlan: guest
acls_in: [access-port-protect]

    4:
      name: "s1"
description: "services Laptop"
native_vlan: guest
acls_in: [access-port-protect]

    24:
      name: "trunk"
description: "VLAN trunk to sw1"
tagged_vlans: [office, guest]

Listing 45: acls.yaml

```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
```
(continues on next page)
# 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:
        - 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.

Configuration options

Top Level

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acls</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration specific to acls. The keys are names of each acl, and the values are config dictionaries holding the acl’s configuration (see below).</td>
</tr>
<tr>
<td>dps</td>
<td>dictionary</td>
<td>{}</td>
<td>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).</td>
</tr>
<tr>
<td>meters</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration specific to meters. The keys are names of each meter, and the values are config dictionaries holding the meter’s configuration (see below).</td>
</tr>
<tr>
<td>routers</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration specific to routers. The keys are names of each router, and the values are config dictionaries holding the router’s configuration (see below).</td>
</tr>
<tr>
<td>version</td>
<td>integer</td>
<td>2</td>
<td>The config version. 2 is the only supported version.</td>
</tr>
<tr>
<td>vlans</td>
<td>dictionary</td>
<td>{}</td>
<td>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).</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>advertise_interval</td>
<td>integer</td>
<td>30</td>
<td>How often to advertise (eg. IPv6 RAs)</td>
</tr>
<tr>
<td>arp_neighbor_timeout</td>
<td>integer</td>
<td>250</td>
<td>ARP and neighbour timeout in seconds</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>name</td>
<td>Description of this datapath, strictly informational</td>
</tr>
<tr>
<td>dot1x</td>
<td>dictionary</td>
<td>{}</td>
<td>802.1X configuration (see below)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dp_id</td>
<td>integer</td>
<td></td>
<td>The configuration key. The OFP datapath-id of this datapath</td>
</tr>
<tr>
<td>drop_bpdu</td>
<td>boolean</td>
<td>True</td>
<td>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 BDPUs or Faucet will forward them as though they were normal traffic.</td>
</tr>
<tr>
<td>drop_broadcast_source_address</td>
<td>boolean</td>
<td>True</td>
<td>If True, Faucet will drop any packet from a broadcast source address</td>
</tr>
<tr>
<td>drop_spoofed_faucet_mac</td>
<td>boolean</td>
<td>True</td>
<td>If True, Faucet will drop any packet it receives with an ethernet source address equal to a MAC address that Faucet is using.</td>
</tr>
<tr>
<td>group_table</td>
<td>boolean</td>
<td>False</td>
<td>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.</td>
</tr>
<tr>
<td>hardware</td>
<td>string</td>
<td>“Open vSwitch”</td>
<td>The hardware model of the datapath. Defaults to “Open vSwitch”. Other options can be seen in the documentation for valve.py</td>
</tr>
<tr>
<td>high_priority</td>
<td>integer</td>
<td>low_priority + 1 (9001)</td>
<td>The high priority value.</td>
</tr>
<tr>
<td>highest_priority</td>
<td>integer</td>
<td>high_priority + 98 (9099)</td>
<td>The highest priority number to use.</td>
</tr>
<tr>
<td>ignore_learn_ins</td>
<td>integer</td>
<td>10</td>
<td>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.</td>
</tr>
<tr>
<td>interfaces</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration block for interface specific config (see below)</td>
</tr>
<tr>
<td>interface_ranges</td>
<td>dictionary</td>
<td>{}</td>
<td>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”.</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>learn_ban_timeout</td>
<td>integer</td>
<td>10</td>
<td>When a host is rapidly moving between ports, Faucet will stop learning MAC addresses on one of the ports for this number of seconds.</td>
</tr>
<tr>
<td>learn_jitter</td>
<td>integer</td>
<td>10</td>
<td>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.</td>
</tr>
<tr>
<td>lldp_beacon</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration block for LLDP beacons</td>
</tr>
<tr>
<td>low_priority</td>
<td>integer</td>
<td>low_priority + 9000 (9000)</td>
<td>The low priority value.</td>
</tr>
<tr>
<td>lowest_priority</td>
<td>integer</td>
<td>priority_offset (0)</td>
<td>The lowest priority number to use.</td>
</tr>
<tr>
<td>max_host_fib_retry_count</td>
<td>integer</td>
<td>10</td>
<td>Limit the number of times Faucet will attempt to resolve a next-hop’s L2 address.</td>
</tr>
<tr>
<td>max_hosts_per_resolve_cycle</td>
<td>integer</td>
<td>5</td>
<td>Limit the number of hosts resolved per cycle.</td>
</tr>
<tr>
<td>max_resolve_backoff_time</td>
<td>integer</td>
<td>32</td>
<td>When resolving next hop L2 addresses, Faucet will back off exponentially until it reaches this value.</td>
</tr>
<tr>
<td>metrics_rate_limit_sec</td>
<td>integer</td>
<td>0</td>
<td>Rate limit metric updates - don’t update metrics if last update was less than this many seconds ago.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td>A name to reference the datapath by.</td>
</tr>
<tr>
<td>ofchannel_log</td>
<td>string</td>
<td>None</td>
<td>Name of logfile for openflow logs</td>
</tr>
<tr>
<td>packetin_pps</td>
<td>integer</td>
<td>None</td>
<td>Ask switch to rate limit packetin in pps.</td>
</tr>
<tr>
<td>slowpath_pps</td>
<td>integer</td>
<td>None</td>
<td>Ask switch to rate limit slowpath in pps.</td>
</tr>
<tr>
<td>priority_offset</td>
<td>integer</td>
<td>0</td>
<td>Shift all priority values by this number.</td>
</tr>
<tr>
<td>proactive_learn_v4</td>
<td>boolean</td>
<td>True</td>
<td>Whether proactive learning is enabled for IPv4 nexthops</td>
</tr>
<tr>
<td>proactive_learn_v6</td>
<td>boolean</td>
<td>True</td>
<td>Whether proactive learning is enabled for IPv6 nexthops</td>
</tr>
<tr>
<td>stack</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration block for stacking config, for loop protection (see below)</td>
</tr>
<tr>
<td>timeout</td>
<td>integer</td>
<td>300</td>
<td>Timeout for MAC address learning</td>
</tr>
<tr>
<td>use_idle_timeout</td>
<td>boolean</td>
<td>False</td>
<td>Turn on/off the use of idle timeout for src_table, default OFF.</td>
</tr>
<tr>
<td>table_sizes</td>
<td>dictionary</td>
<td>{}</td>
<td>For TFM based switches, size of each FAUCET table (any may be specified)</td>
</tr>
<tr>
<td>port_table_scale_factor</td>
<td>float</td>
<td>1.0</td>
<td>For TFM based switches, and for tables that are sized by number of ports, scale size estimate.</td>
</tr>
<tr>
<td>global_vlan</td>
<td>int</td>
<td>2**11-1</td>
<td>When global routing is enabled, FIB VID used internally</td>
</tr>
</tbody>
</table>
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 within the configuration block ‘stack’:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>integer</td>
<td>0</td>
<td>Setting any value for stack priority indicates that this datapath should be the root for the stacking topology.</td>
</tr>
</tbody>
</table>

LLDP (DP)

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

LLDP beacons can be used to, among other things, facilitate physical troubleshooting (e.g. so that a standard cable tester can display port information), verify FAUCET stacking topology, and cue a phone to use the right voice VLAN.

**Note:** While FAUCET can receive and log LLDP from other devices, FAUCET does not do spanning tree. Those LLDP packets will have no influence on FAUCET’s forwarding decisions.

The following attributes can be configured within the ‘lldp_beacon’ configuration block at the dp level:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system_name</td>
<td>string</td>
<td>The datapath name</td>
<td>System name inside LLDP packet</td>
</tr>
<tr>
<td>send_interval</td>
<td>integer</td>
<td>None</td>
<td>Seconds between sending beacons</td>
</tr>
<tr>
<td>max_per_interval</td>
<td>integer</td>
<td>None</td>
<td>The maximum number of beacons, across all ports to send each interval</td>
</tr>
</tbody>
</table>

**Note:** When stack ports are enabled FAUCET automatically configures LLDP with the default values for send_interval and max_per_interval to 5.

802.1X (DP)

**Note:** 802.1X support is experimental, and there may be incomplete features or bugs. If you find an issue please email the mailing list or create an Github issue.

Faucet implements 802.1X by forwarding EAPOL packets on the dataplane to a socket it is listening on. These packets are then passed through to a RADIUS server which performs the authentication and generates the reply message.

For each instance of Faucet there is only one 802.1X speaker. This 802.1X speaker is configured by the options below. Except for the ‘nfv_sw_port’ option, the configuration for the speaker is configured using the first dp’s dot1x config dictionary. For all other dps only the ‘nfv_sw_port’ option is required with the others ignored.

A basic network and configuration with two hosts may look like:
Faucet

Controlplane

Dataplane

dps:

dp1:

  nfv_intf: eth1
  nfv_sw_port: 4
  ...

H1

H2

1.4. Configuration
A brief overview of the current state of the implementation:

Implemented:

- EAP Types: MD5, PEAP, TLS, TTLS.
- Authentication session expiry default 3600 seconds. (configurable (per authentication) via returning the Session-Timeout attribute in the RADIUS Access-Accept message).
- Faucet connects to a single RADIUS server, and passes through all EAP messages.
- Client can end session with EAP-Logoff.
- Dynamic assignment of the native VLAN. Use RADIUS attribute Private-Group-Tunnel-ID in Radius Access-Accept with the name of the faucet VLAN.

Not Supported (yet):

- RADIUS Accounting.
- Multiple RADIUS Servers.
- Other EAP types. E.g. FAST, …
- Dynamic assignment of ACL.

802.1X port authentication is configured in the dp configuration block and in the interface configuration block. At the dp level the following attributes can be configured with the configuration block ‘dot1x’:

Table 5: dps: <dp name or id>: dot1x: {}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nfv_intf</td>
<td>str</td>
<td></td>
<td>The interface for Faucet to listen for EAP packets from the dataplane. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>nfv_sw_port</td>
<td>int</td>
<td></td>
<td>Switch port number that connects to the Faucet server’s nfv_intf</td>
</tr>
<tr>
<td>radius_ip</td>
<td>str</td>
<td></td>
<td>IP address of RADIUS Server the 802.1X speaker will authenticate with. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>radius_port</td>
<td>int</td>
<td>1812</td>
<td>UDP port of RADIUS Server the 802.1X speaker will authenticate with. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>radius_secret</td>
<td>str</td>
<td></td>
<td>Shared secret used by the RADIUS server and the 802.1X speaker. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>noauth_acl</td>
<td>str</td>
<td>None</td>
<td>The name of the defined ACL [refer to acls.yaml for more information] that will be set to all 802.1X ports by default, that is before any user is authenticated. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>auth_acl</td>
<td>str</td>
<td>None</td>
<td>The name of the defined ACL [refer to acls.yaml for more information] that will be set to an 802.1X port when a user authenticates. - NOTE: Faucet will only use the config from the first dp</td>
</tr>
<tr>
<td>dot1x_assigned</td>
<td>boolean</td>
<td>False</td>
<td>True, if this ACL can be dynamically assigned by a RADIUS server during 802.1X authentication.</td>
</tr>
</tbody>
</table>

**Interfaces**

Configuration for each interface is entered in the ‘interfaces’ configuration block within 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 6: dps: <dp name or id>: interfaces: <interface name or OFP port number>: {}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_in</td>
<td>integer or string</td>
<td>None</td>
<td>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</td>
</tr>
<tr>
<td>acls_in</td>
<td>a list of ACLs, as integers or strings</td>
<td>None</td>
<td>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.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Name</td>
<td>Description, purely informational</td>
</tr>
<tr>
<td>dot1x</td>
<td>boolean</td>
<td>False</td>
<td>Enable 802.1X port authentication (NOTE: Requires ACL with dot1x_assigned attribute, for 802.1X Per User ACLs)</td>
</tr>
<tr>
<td>dot1x_acl</td>
<td>boolean</td>
<td>False</td>
<td>Enable 802.1X ACL functionality on port (NOTE: Requires dot1x attribute)</td>
</tr>
<tr>
<td>dot1x_mab</td>
<td>boolean</td>
<td>False</td>
<td>Enable 802.1X Mac Authentication Bypass on port (NOTE: Requires dot1x attribute)</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>True</td>
<td>Allow packets to be forwarded through this port.</td>
</tr>
<tr>
<td>hairpin</td>
<td>boolean</td>
<td>False</td>
<td>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.</td>
</tr>
<tr>
<td>lldp_beacon</td>
<td>dictionary</td>
<td>{}</td>
<td>Configuration block for lldp configuration</td>
</tr>
<tr>
<td>loop_protect</td>
<td>boolean</td>
<td>False</td>
<td>if True, do simple (host/access port) loop protection on this port.</td>
</tr>
<tr>
<td>loop_protect_external</td>
<td>boolean</td>
<td>False</td>
<td>if True, do external (other switch) loop protection on this port.</td>
</tr>
<tr>
<td>max_hosts</td>
<td>integer</td>
<td>255</td>
<td>the maximum number of mac addresses that can be learnt on this port.</td>
</tr>
<tr>
<td>mirror</td>
<td>a list of integers or strings</td>
<td>None</td>
<td>Mirror all allowed packets recieved from (subject to ACLs), and all packets transmitted to, the ports specified (by name or by port number), to this port. If mirroring of denied by ACL packets is desired, use the ACL rule mirror option. The mirrored packets are from the perspective of hosts on the mirrored port (for example, a packet with a VLAN tag, transmitted to a host on a mirrored and untagged port, will be mirrored without its original VLAN tag). NOTE: If packets are exchanged between two ports that are both mirrored, depending on the OpenFlow switch, only one copy of the mirrored traffic may be sent (when a port sends a packet, not when the other receives it). This is because some implementations cannot send a packet more than once to the same port.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The configuration key.</td>
<td></td>
</tr>
<tr>
<td>native_vlan</td>
<td>integer</td>
<td>None</td>
<td>The vlan associated with untagged packets arriving and leaving this interface.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp</td>
<td>integer or string</td>
<td>None</td>
<td>The name or dp_id of the dp connected to this port</td>
</tr>
<tr>
<td>port</td>
<td>integer or string</td>
<td>None</td>
<td>The name or OFP port number of the interface on the remote dp connected to this interface.</td>
</tr>
</tbody>
</table>

LLDP (Interfaces)

Interface specific configuration for LLDP.

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

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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>string</td>
<td>None</td>
<td>The info field of the tlv, as a hex string</td>
</tr>
<tr>
<td>oui</td>
<td>integer</td>
<td>None</td>
<td>The Organisationally Unique Identifier</td>
</tr>
<tr>
<td>subtype</td>
<td>integer</td>
<td>None</td>
<td>The organizationally defined subtype</td>
</tr>
</tbody>
</table>
Router

Routers config is used to allow routing between VLANs, and optionally BGP. 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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlans</td>
<td>list of integers or strings</td>
<td>None</td>
<td>Enables inter-vlan routing on the given VLANs.</td>
</tr>
<tr>
<td>bgp</td>
<td>BGP configuration.</td>
<td>None</td>
<td>See below for BGP configuration.</td>
</tr>
</tbody>
</table>

BGP

Routers config to enable BGP routing.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>integer</td>
<td>None</td>
<td>The local AS number to used when speaking BGP</td>
</tr>
<tr>
<td>connect_mode</td>
<td>string</td>
<td>“passive”</td>
<td>Must be “passive”</td>
</tr>
<tr>
<td>neighbor_addresses</td>
<td>list of strings (IP addresses)</td>
<td>None</td>
<td>The list of BGP neighbours</td>
</tr>
<tr>
<td>neighbor_as</td>
<td>integer</td>
<td>None</td>
<td>The AS Number for the BGP neighbours</td>
</tr>
<tr>
<td>routerid</td>
<td>string (IP address)</td>
<td>None</td>
<td>BGP router ID.</td>
</tr>
<tr>
<td>server_addresses</td>
<td>list of strings (IP addresses)</td>
<td>None</td>
<td>IP addresses for FAUCET to listen for incoming BGP addresses.</td>
</tr>
<tr>
<td>port</td>
<td>integer</td>
<td>None</td>
<td>Port to use for BGP sessions</td>
</tr>
<tr>
<td>vlan</td>
<td>string</td>
<td>None</td>
<td>The VLAN to add/remove BGP routes from.</td>
</tr>
</tbody>
</table>

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 12: vlans: <vlan name or vid>: {}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_in</td>
<td>string or integer</td>
<td>None</td>
<td>Deprecated, replaced by acls_in which accepts a list. The acl to be applied to all packets arriving on this vlan.</td>
</tr>
<tr>
<td>acls_in</td>
<td>a list of ACLs, as integers or strings</td>
<td>None</td>
<td>The acl to be applied to all packets arriving on this vlan. ACLs listed first take priority over those later in the list.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>None</td>
<td>Strictly informational</td>
</tr>
<tr>
<td>dot1x_assigned</td>
<td>bool</td>
<td>False</td>
<td>True, if this VLAN can be dynamically assigned by a RADIUS server during 802.1X authentication. Otherwise False</td>
</tr>
<tr>
<td>faucet_vips</td>
<td>list of strings (IP address prefixes)</td>
<td>None</td>
<td>The IP Address for Faucet’s routing interface on this vlan</td>
</tr>
<tr>
<td>faucet_mac</td>
<td>string (MAC address)</td>
<td>None</td>
<td>Set MAC for FAUCET VIPs on this VLAN</td>
</tr>
<tr>
<td>max_hosts</td>
<td>integer</td>
<td>255</td>
<td>The maximum number of hosts that can be learnt on this vlan.</td>
</tr>
<tr>
<td>minimum_ip_size_check</td>
<td>boolean</td>
<td>True</td>
<td>If False, don’t check that IP packets have a payload (must be False for OVS trace/tutorial to work)</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>the configuration key</td>
<td>A name that can be used to refer to this vlan.</td>
</tr>
<tr>
<td>proactive_arp_limit</td>
<td>integer</td>
<td>2052</td>
<td>Do not proactively ARP for hosts once this value has been reached (set to None for unlimited)</td>
</tr>
<tr>
<td>proactive_nd_limit</td>
<td>integer</td>
<td>2052</td>
<td>Don’t proactively discover IPv6 hosts once this value has been reached (set to None for unlimited)</td>
</tr>
<tr>
<td>routes</td>
<td>list of routes</td>
<td>None</td>
<td>Static routes configured on this vlan (see below)</td>
</tr>
<tr>
<td>targeted_gw_resolution</td>
<td>boolean</td>
<td>False</td>
<td>If True, and a gateway has been resolved, target the first re-resolution attempt to the same port rather than flooding.</td>
</tr>
<tr>
<td>unicast_flood</td>
<td>boolean</td>
<td>True</td>
<td>If False packets to unknown ethernet destination MAC addresses will be dropped rather than flooded.</td>
</tr>
<tr>
<td>vid</td>
<td>integer</td>
<td>the configuration key</td>
<td>The vid for the vlan.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_dst</td>
<td>string (IP subnet)</td>
<td>None</td>
<td>The destination subnet.</td>
</tr>
<tr>
<td>ip_gw</td>
<td>string (IP address)</td>
<td>None</td>
<td>The next hop for this route</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meter_id</td>
<td>int</td>
<td></td>
<td>Unique identifier.</td>
</tr>
<tr>
<td>entry</td>
<td>dictionary</td>
<td></td>
<td>Defines the meter actions. Details Below.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>string or list of strings</td>
<td>KBPS</td>
<td>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)</td>
</tr>
<tr>
<td>bands</td>
<td>list of bands (which are dictionaries, see below)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 16: : meters: <meter name>: entry: bands:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
<td>'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.</td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td>int</td>
<td>Rate for dropping or remarking packets, depending on the above type. Value is in KBPS or PKTPS flag depending on the flag set.</td>
<td></td>
</tr>
<tr>
<td>burst_size</td>
<td>int</td>
<td>Only used if flags includes BURST. Indicates the length of packet or byte burst to consider for applying the meter.</td>
<td></td>
</tr>
<tr>
<td>prec_level</td>
<td>int</td>
<td>Only used if type is DSCP_REMARK. The amount by which the drop precedence should be increased.</td>
<td></td>
</tr>
</tbody>
</table>

**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 17: : acls: <acl name>: - rule: actions: {}:

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

The output action contains a dictionary with the following elements:
### Table 18: : acls: <acl name>: - rule: actions: output: {}

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

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

### Table 19: : acls: <acl name>: - rule: actions: output: failover: {}

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>integer</td>
<td>None</td>
<td>The OFP group id to use for the failover group</td>
</tr>
<tr>
<td>ports</td>
<td>list</td>
<td>None</td>
<td>The list of ports the packet can be output through.</td>
</tr>
</tbody>
</table>

### 1.4.2 Gauge configuration

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

Listing 46: gauge.yaml

```yaml
# Recommended configuration is Prometheus for all monitoring, with all_dps: True
faucet_configs:
- '/etc/faucet/faucet.yaml'
watches:
  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'
```

(continues on next page)
1.4.3 Verifying configuration

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

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

1.4.4 Configuration examples

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

1.4.5 Applying configuration updates

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

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

1.4.6 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
```
**1.4.7 Environment variables**

You can use environment variables to override default behaviour of faucet such as paths for configuration files and port numbers.
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAUCET_CONFIG</td>
<td>Colon-separated list of file paths</td>
<td>/etc/faucet/faucet.yaml: /etc/ryu/faucet/faucet.yaml</td>
<td>Faucet will load its configuration from the first valid file in list</td>
</tr>
<tr>
<td>FAUCET_CONFIG_AUTO_REVERT</td>
<td>boolean</td>
<td>False</td>
<td>If true, Faucet will attempt to revert a bad config file back to the last known good version.</td>
</tr>
<tr>
<td>FAUCET_CONFIG_STAT_RELOAD</td>
<td>boolean</td>
<td>False</td>
<td>If true, faucet will automatically reload itself and apply new configuration when FAUCET_CONFIG changes</td>
</tr>
<tr>
<td>FAUCET_LOG_LEVEL</td>
<td>Python log level</td>
<td>INFO</td>
<td>Log verbosity</td>
</tr>
<tr>
<td>FAUCET_LOG</td>
<td>File path or STDOUT or STDERR</td>
<td>/var/log/faucet/ faucet.log</td>
<td>Location for faucet to log messages to, can be special values STDOUT or STDERR</td>
</tr>
<tr>
<td>FAUCET_EXCEPTION_LOG</td>
<td>File path or STDOUT or STDERR</td>
<td>/var/log/faucet/ gauge_exception.log</td>
<td>Location for faucet log to log exceptions to, can be special values STDOUT or STDERR</td>
</tr>
<tr>
<td>FAUCET_EVENT_SOCK</td>
<td>Socket path</td>
<td></td>
<td>Location to a UNIX socket where faucet will write events to, or empty to disable events</td>
</tr>
<tr>
<td>FAUCET_EVENT_SOCK_HEARTBEAT</td>
<td>Seconds</td>
<td>0</td>
<td>If set to a value greater than 0, it emits a dummy event every n seconds so that faucet knows if the event socket connection is broken and closes the connection on it’s side.</td>
</tr>
<tr>
<td>FAUCET_PROMETHEUS_PORT</td>
<td>Port</td>
<td>9302</td>
<td>TCP port to listen on for faucet prometheus client</td>
</tr>
<tr>
<td>FAUCET_PROMETHEUS_ADDR</td>
<td>IP address</td>
<td>0.0.0.0</td>
<td>IP address to listen on for faucet prometheus client</td>
</tr>
<tr>
<td>GAUGE_CONFIG</td>
<td>Colon-separated list of file paths</td>
<td>/etc/faucet/gauge.yaml: /etc/ryu/faucet/gauge.yaml</td>
<td>Gauge will load it’s configuration from the first valid file in list</td>
</tr>
<tr>
<td>GAUGE_CONFIG_STAT_RELOAD</td>
<td>boolean</td>
<td>False</td>
<td>If true, gauge will automatically reload itself and apply new configuration when GAUGE_CONFIG changes</td>
</tr>
<tr>
<td>GAUGE_LOG_LEVEL</td>
<td>Python log level</td>
<td>INFO</td>
<td>Log verbosity</td>
</tr>
<tr>
<td>GAUGE_LOG</td>
<td>File path or STDOUT or STDERR</td>
<td>/var/log/faucet/ gauge.log</td>
<td>Location for gauge to log messages to, can be special values STDOUT or STDERR</td>
</tr>
<tr>
<td>GAUGE例外LOG</td>
<td>File path or STDOUT or STDERR</td>
<td>/var/log/faucet/ gauge_exception.log</td>
<td>Location for faucet log to log exceptions to, can be special values STDOUT or STDERR</td>
</tr>
</tbody>
</table>
1.5 Monitoring

Faucet can be monitored in a number of ways. Both the faucet and gauge services export instrumentation data via a built-in Prometheus exporter which can be consumed by Prometheus. By default the Prometheus exporter is available on port 9302, this can be changed with Environment variables (FAUCET_PROMETHEUS_PORT and FAUCET_PROMETHEUS_ADDR).

Gauge also collects conventional switch statistics (port counters, port state, etc) and can export these to a number of different databases (including Prometheus). For information on configuring gauge see the Gauge configuration section.

1.5.1 Prometheus metrics

Below is a list of the metrics exported by faucet and gauge.

Exported by faucet

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet_pbr_version</td>
<td>gauge</td>
<td>Faucet PBR version</td>
</tr>
<tr>
<td>faucet_stack_root_dpid</td>
<td>gauge</td>
<td>set to current stack root DPID</td>
</tr>
<tr>
<td>faucet_config_reload_requests_total</td>
<td>counter</td>
<td>number of config reload requests</td>
</tr>
<tr>
<td>faucet_config_load_error</td>
<td>gauge</td>
<td>1 if last attempt to re/load config failed</td>
</tr>
<tr>
<td>faucet_config_hash</td>
<td>info</td>
<td>file hashes for last successful config</td>
</tr>
<tr>
<td>faucet_config_hash_func</td>
<td>gauge</td>
<td>algorithm used to compute config hashes</td>
</tr>
<tr>
<td>faucet_config_applied</td>
<td>gauge</td>
<td>fraction of DPs that we have tried to apply config to</td>
</tr>
<tr>
<td>faucet_event_id</td>
<td>gauge</td>
<td>highest/most recent event ID to be sent</td>
</tr>
<tr>
<td>faucet_config_reload_warm_total</td>
<td>counter</td>
<td>number of warm, differences only config reloads executed</td>
</tr>
<tr>
<td>faucet_config_reload_cold_total</td>
<td>counter</td>
<td>number of cold, complete reprovision config reloads executed</td>
</tr>
<tr>
<td>of_ignored_packet_ins_total</td>
<td>counter</td>
<td>number of OF packet_ins received but ignored from DP (due to rate limiting)</td>
</tr>
<tr>
<td>of_unexpected_packet_ins_total</td>
<td>counter</td>
<td>number of OF packet_ins received that are unexpected from DP (e.g. for unknown VLAN)</td>
</tr>
<tr>
<td>of_packet_ins_total</td>
<td>counter</td>
<td>number of OF packet_ins received from DP</td>
</tr>
<tr>
<td>of_non_vlan_packet_ins_total</td>
<td>counter</td>
<td>number of OF packet_ins received from DP, not associated with a FAUCET VLAN</td>
</tr>
<tr>
<td>of_vlan_packet_ins_total</td>
<td>counter</td>
<td>number of OF packet_ins received from DP, associated with a FAUCET VLAN</td>
</tr>
<tr>
<td>of_flowmsgs_sent_total</td>
<td>counter</td>
<td>number of OF flow messages (and packet outs) sent to DP</td>
</tr>
<tr>
<td>of_errors_total</td>
<td>counter</td>
<td>number of OF errors received from DP</td>
</tr>
<tr>
<td>of_dp_connections_total</td>
<td>counter</td>
<td>number of OF connections from a DP</td>
</tr>
<tr>
<td>of_dp_disconnections_total</td>
<td>counter</td>
<td>number of OF connections from a DP</td>
</tr>
<tr>
<td>vlan_hosts_learned</td>
<td>gauge</td>
<td>number of hosts learned on a VLAN</td>
</tr>
<tr>
<td>port_vlan_hosts_learned</td>
<td>gauge</td>
<td>number of hosts learned on a port and VLAN</td>
</tr>
<tr>
<td>vlan_neighbors</td>
<td>gauge</td>
<td>number of L3 neighbors on a VLAN (whether resolved to L2 addresses, or not)</td>
</tr>
<tr>
<td>vlan_learn_bans</td>
<td>gauge</td>
<td>number of times learning was banned on a VLAN</td>
</tr>
<tr>
<td>faucet_config_table_names</td>
<td>gauge</td>
<td>number to names map of FAUCET pipeline tables</td>
</tr>
</tbody>
</table>

Continued on next page
Table 20 – continued from previous page

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet_packet_in_secs</td>
<td>histogram</td>
<td>FAUCET packet in processing time</td>
</tr>
<tr>
<td>faucet_valve_service_secs</td>
<td>histogram</td>
<td>FAUCET valve service processing time</td>
</tr>
<tr>
<td>bgp_neighbor_upptime</td>
<td>gauge</td>
<td>BGP neighbor uptime in seconds</td>
</tr>
<tr>
<td>bgp_neighbor_routes</td>
<td>gauge</td>
<td>BGP neighbor route count</td>
</tr>
<tr>
<td>learned_macs</td>
<td>gauge</td>
<td>MAC address stored as 64bit number to DP ID, port, VLAN, and n (discrete index)</td>
</tr>
<tr>
<td>port_status</td>
<td>gauge</td>
<td>status of switch ports</td>
</tr>
<tr>
<td>port_stack_state</td>
<td>gauge</td>
<td>state of stacking on a port</td>
</tr>
<tr>
<td>port_learn_bans</td>
<td>gauge</td>
<td>number of times learning was banned on a port</td>
</tr>
<tr>
<td>learned_l2_port</td>
<td>gauge</td>
<td>learned port of l2 entries</td>
</tr>
<tr>
<td>port_lacp_state</td>
<td>gauge</td>
<td>state of LACP on a port</td>
</tr>
<tr>
<td>dp_status</td>
<td>gauge</td>
<td>status of datapaths</td>
</tr>
<tr>
<td>dp_root_hop_port</td>
<td>gauge</td>
<td>port that leads to stack root DP</td>
</tr>
<tr>
<td>of_dp_desc_stats</td>
<td>gauge</td>
<td>DP description (OFPDescStatsReply)</td>
</tr>
<tr>
<td>stack_cabling_errors_total</td>
<td>counter</td>
<td>number of cabling errors detected in all FAUCET stacks</td>
</tr>
<tr>
<td>stack_probes_received_total</td>
<td>counter</td>
<td>number of stacking messages received</td>
</tr>
<tr>
<td>dp_dot1x_success_total</td>
<td>counter</td>
<td>number of successful authentications on dp</td>
</tr>
<tr>
<td>dp_dot1x_failure_total</td>
<td>counter</td>
<td>number of authentications attempts failed on dp</td>
</tr>
<tr>
<td>dp_dot1x_logoff_total</td>
<td>counter</td>
<td>number of eap-logoff events on dp</td>
</tr>
<tr>
<td>port_dot1x_success_total</td>
<td>counter</td>
<td>number of successful authentications on port</td>
</tr>
<tr>
<td>port_dot1x_failure_total</td>
<td>counter</td>
<td>number of authentications attempts failed on port</td>
</tr>
<tr>
<td>port_dot1x_logoff_total</td>
<td>counter</td>
<td>number of eap-logoff events on port</td>
</tr>
</tbody>
</table>

Exported by gauge

Table 21: Gauge prometheus metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet_pbr_version</td>
<td>gauge</td>
<td>Faucet PBR version</td>
</tr>
<tr>
<td>dp_status</td>
<td>gauge</td>
<td>status of datapaths</td>
</tr>
<tr>
<td>of_port_tx_packets</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_rx_packets</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_tx_bytes</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_rx_bytes</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_tx_dropped</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_rx_dropped</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_tx_errors</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_rx_errors</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_reason</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_state</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_portCurr_speed</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_port_max_speed</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_meter_flow_count</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_meter_byte_in_count</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_meter_packet_in_count</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_meter_byte_band_count</td>
<td>gauge</td>
<td></td>
</tr>
<tr>
<td>of_meter_packet_band_count</td>
<td>gauge</td>
<td></td>
</tr>
</tbody>
</table>
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 10/18/28/52/96 with POE+ models as well. Here is a list of some of our most popular switches:

- AT-x930
- SBx908Gen2
- AT-x950
- 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.

```bash
/* 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
```

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

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

/* NOTE - Starting from AlliedWarePlus version 5.4.8-2, we have added support for controller name. */
    * You can specify a controller name with the optional <name> parameter. */
    * Users can still use the previous controller commands (without the name parameter) and the switch will auto-generate a suitable name (starting with "oc") in that case. */
    * Here is an example to add a controller with name 'faucet' using TCP port 6653 */
    awplus (config)# openflow controller faucet tcp 192.168.1.10 6653

/* 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

/* Disable Loop protection detection Globally */
    awplus (config)# no loop-protection loop-detect

/* 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) */
Faucet Documentation

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! */
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 data VLAN(s) must be included in the vlan database config
 * on the switch before they can be used by OpenFlow.
 * Here is an example to create DP VLANs 2-100 */
awplus (config)# vlan database
awplus (config-vlan)# vlan 2-100

/* NOTE - Starting from software version 5.4.8-2, in order to negate a controller, you need to specify the controller name.
 * In case you add the controller the legacy way (without the name), the newer software will auto-generate a name which can be
 * used to delete the controller.
 * Here is an example to delete a controller with auto-generated name ocl */
awplus (config)# no openflow controller ocl

Useful Switch related configurations

Note: If the Openflow controller is located in a different VLAN or Network segment, routing needs to be configured so that the switch can talk to the controller.

/* To set Timezone: Codes - https://www.timeanddate.com/time/zones/ */
/* For US Pacific Time zone */
awplus (config)# clock timezone NAPST minus 8

/* To set DNS, say a local Gateway also acting as a DNS forwarder 10.20.0.1 */
awplus (config)# ip name-server 10.20.0.1

/* To make sure that DNS and routing correctly work, Gateway address needs to be set. */
/* Here, Gateway is set only to the management VLAN, vlan1; 255 is the max depth allowed */
awplus (config)# ip route 0.0.0.0/0 vlan1 255
awplus (config)# ip route 0.0.0.0/0 10.20.0.1

/* To see the configured Route database */
awplus# show ip route database

/* To test routing, ping Google.com - note the name to ip resolution */
awplus# ping google.com

Setting up PKI Certs for secure connectivity between Switch and Openflow Controller

Note: There are many ways to get the keys and certificates into the box. Here, both private key (unencrypted PEM formatted) and corresponding Certificate (PEM) as trusted by the Openflow Controller is provided to the Switch Admin
for installation.

Getting keys into the Switch flash partition

```bash
awplus# copy scp://user@10.20.5.5/home/user/switch-cert.pem switch-cert.pem
awplus# copy scp://user@10.20.5.5/home/user/switch-key_nopass.pem switch-key_nopass.pem

/* Showing only relevant files */
awplus# dir
  1679 -rw- Dec 20 2017 09:04:35 switch-key_nopass.pem
  11993 -rw- Dec 20 2017 09:04:03 switch-cert.pem
```

Setting up Trustpoint for SSL connectivity to Openflow Controller

```bash
awplus (config)# crypto pki trustpoint local

/* Point the switch to the OF controller */
awplus (config)# openflow controller ssl 192.168.1.10 6653

/* Allow OpenFlow to use local trustpoint */
awplus (config)# openflow ssl trustpoint local

/* Copy the new key and pvt keys to local trustpoint directory */
awplus# copy switch-key_nopass.pem .certs/pki/local/cakey.pem
Overwrite flash:/.certs/pki/local/cakey.pem (y/n)[n]:y
Copying...
Successful operation

awplus# copy switch-cert.pem .certs/pki/local/cacert.pem
Overwrite flash:/.certs/pki/local/cacert.pem (y/n)[n]:y
Copying...
Successful operation
```

Enabling SNMP for monitoring Management/Control Plane Port

Openflow enabled ports are monitored via Openflow Stats request/response protocol. This means that Management port (and if Openflow control channel port is separate), are not monitored on the switch. Hence, SNMP is used to monitor the same. SNMP v2 is the most widely used. As an example below, let us assume NMS is @ 10.20.30.71

```bash
/* Check contents of existing access-list */
awplus# show access-list

/* Enable the SNMP agent and enable the generation of authenticate */
* failure traps to monitor unauthorized SNMP access. */
awplus (config)# snmp-server enable trap auth

/* Creating a write access community called sfractalonpremlrw for use by */
* the central network management station at 10.20.30.71 */
awplus (config)# access-list 96 permit 10.20.30.71
awplus (config)# snmp-server community sfractalonpremlrw rw view atview 96

/* Enable link traps on VLANs or specific interfaces (in our case management port) */
```
Faucet Documentation

awplus (config)# interface port1.0.1
awplus (config-if)# snmp trap link-status

/* Configuring Trap Hosts */
awplus (config)# snmp-server host 10.20.30.71 version 2c sfractalonprem1rw

/* Confirm all SNMP settings */
awplus# show snmp-server
SNMP Server .......................... Enabled
IP Protocol .......................... IPv4, IPv6
SNMP Startup Trap Delay .............. 30 Seconds
SNMPv3 Engine ID (configured name) ... Not set
SNMPv3 Engine ID (actual) ............ 0x80001f8880a2977c410e3bb658

awplus# show snmp-server community
SNMP community information:
Community Name ........... sfractalonprem1rw
Access .................. Read-write
View .................. atview

awplus# show run snmp
snmp-server
snmp-server enable trap auth
snmp-server community sfractalonprem1rw rw view atview 96
snmp-server host 10.20.30.71 version 2c sfractalonprem1rw
!

/* Check if the interface is configured for SNMP */
awplus# show interface port1.0.1
Interface port1.0.1
  Scope: both
  Link is UP, administrative state is UP
  Thrash-limiting
    Status Not Detected, Action learn-disable, Timeout 1(s)
  Hardware is Ethernet, address is 001a.eb96.6ef2
  index 5001 metric 1 mru 1500
  current duplex full, current speed 1000, current polarity mdi
  configured duplex auto, configured speed auto, configured polarity auto
  <UP,BROADCAST,RUNNING,MULTICAST>
  SNMP link-status traps: Sending (suppressed after 20 traps in 60 sec)
    Link-status trap delay: 0 sec
      input packets 14327037, bytes 3727488153, dropped 0, multicast packets 440768
      output packets 11172202, bytes 2028940085, multicast packets 233192 broadcast,
      packets 1889
    Time since last state change: 40 days 00:48:38

awplus# show access-list
Standard IP access list 96
  10 permit 10.20.30.71

Enabling sFlow for monitoring Management/Control Port

Openflow enabled ports are monitored via Openflow Stats request/response protocol. This means that Management port (and if Openflow control channel port is separate), are not monitored on the switch. Hence, sFlow is used to monitor the same. At this time, no TLS/SSL support is seen on the sFlow Controller channel.
/* Check for any existing sFlow configuration */
awplus# show running-config sflow
!

/* Enable sFlow globally */
awplus (config)# sflow enable
% INFO: sFlow will not function until collector address is non-zero
% INFO: sFlow will not function until agent address is set
awplus# show running-config sflow
!
sflow enable
!

/* Confirm the new sFlow settings */
awplus# show sflow
sFlow Agent Configuration: Default Values
  sFlow Admin Status ........ Enabled [Disabled]
  sFlow Agent Address ....... [not set] [not set]
  Collector Address ......... 0.0.0.0 [0.0.0.0]
  Collector UDP Port ........ 6343 [6343]
  Tx Max Datagram Size ...... 1400 [1400]

sFlow Agent Status:
  Polling/sampling/Tx ........ Inactive because:
    - Agent Addr is not set
    - Collector Addr is 0.0.0.0
    - Polling & sampling disabled on all ports

/* Agent IP MUST be the IP address of the management port of this switch */
awplus (config)# sflow agent ip 192.0.2.23

/* Default sFlow UDP collector port is 6343 */
awplus (config)# sflow collector ip 192.0.2.25 port 6343
awplus (config)# interface port1.0.1
awplus (config-if)# sflow polling-interval 120
awplus (config-if)# sflow sampling-rate 512

awplus# show running-config sflow
!
sflow agent ip 192.0.2.23
sflow collector ip 192.0.2.25
sflow enable
!
interface port1.0.1
  sflow polling-interval 120
  sflow sampling-rate 512
!
awplus#
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.

```yaml
dps:
  allied-telesis:
    dp_id: 0x0000eccd6d123456
    hardware: "Allied-Telesis"
    interfaces:
      1:
        native_vlan: 100
        name: "port1.0.1"
      2:
        name: "port1.0.2"
        description: "windscale"
```

References

- Allied Telesis x930
- OpenFlow Configuration Guide
- Chapter 61 (SNMP)
- SNMP Feature 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. HPE Aruba recommends use of the latest available firmware, which can be downloaded from HPE Support.

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

- IPv6 management of the switch, together OpenFlow is not supported.
- The `OFPAT_DEC_NW_TTL` action is not supported (when FAUCET is configured as a router, IP TTL will not be decremented).

Setup

In all configuration examples following, substitute 10.0.0.1 for your controller IP address, and 10.0.0.2 for your switch IP address, as appropriate. VLAN 2048 is used for the control plane - you can substitute this for another VID. In any case, the control plane VLAN VID you reserve cannot be used in FAUCET’s configuration file (i.e. it cannot be controlled by OpenFlow).

Switch

Chassis configuration (5400R only)

Skip this step if you have a fixed configuration system (2930 or 3810).

On a chassis system with insertable cards, new cards are configured to work in a backwards-compatible way (with reduced functionality) unless older cards are disabled in the chassis. To disable older (V2) cards and enable all functionality necessary to operate FAUCET, put the chassis into a mode where only V3 cards are allowed.

```plaintext
// Disable backwards compatibility, enable full Openflow flexibility
switch (config)# no allow-v2-modules
```

VLAN/port configuration

Aruba switches require the reservation of each VLAN VID you wish to use in FAUCET, on the switch. Some Aruba switches will allow you to reserve a large range of VIDs at once. If your switch has limited resources, then reserve just the VIDs you need.

The reservation of a VID is accomplished by defining a tagged VLAN. Note even you are using that VLAN VID untagged on a port in FAUCET, it must be reserved as tagged on the switch

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

  ```plaintext
  // Increase the maximum number of allowed VLANs on the box and save the configuration.
  // If the switch cannot reserve the full range, reserve only the maximum you need.
  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 10.0.0.2/24

  // Create maximum number of VLANs and tag every dataplane port available to each vlan. Takes up to 30 minutes.
  // If the switch cannot reserve the full range, reserve only the VLANs needed individually.
  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.  
// If the switch cannot reserve the full range, reserve only the maximum you need.  
switch (config)# max-vlans 2048  
switch (config)# write mem  

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

// Create a control-plane vlan and add a single control-plane port (port 48)  
switch (config)# vlan 2048 untagged 48  
switch (config)# vlan 2048 ip address 10.0.0.2/24  

// 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.  
// If the switch cannot reserve the full range, reserve only the VLANs needed  
→ individually.  
switch (config)# vlan 2-2047 tagged 1-47,49-52  

OpenFlow configuration  

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

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

// Enter OpenFlow context  
switch (config)# openflow  

// Configure an OpenFlow controller connection for FAUCET over tcp-port 6653  
switch(openflow)# controller-id 1 ip 10.0.0.1 port 6653 controller-interface oobm  

// Configure an OpenFlow controller connection for Gauge over tcp-port 6654  
switch(openflow)# controller-id 2 ip 10.0.0.1 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 10.0.0.1 port 6653 controller-interface vlan 2048  

// Configure an OpenFlow controller connection for Gauge over tcp-port 6654  
switch(openflow)# controller-id 2 ip 10.0.0.1 port 6654 controller-interface vlan 2048  

// 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  

(continues on next page)
// 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
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.

NOTE: following is an example only, and may look different depending on FAUCET version and which FAUCET features have been enabled.

switch# show openflow instance aggregate flow-table

OpenFlow Instance Flow Table Information

<table>
<thead>
<tr>
<th>Table ID</th>
<th>Table Name</th>
<th>Flow Count</th>
<th>Miss Count</th>
<th>Goto Table</th>
</tr>
</thead>
</table>

(continues on next page)
Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Table Name</th>
<th>Available Free Flow Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Port ACL</td>
<td>Ports 1-52: 46</td>
</tr>
<tr>
<td>1</td>
<td>VLAN</td>
<td>Ports 1-52: 91</td>
</tr>
<tr>
<td>2</td>
<td>VLAN ACL</td>
<td>Ports 1-52: 50</td>
</tr>
<tr>
<td>3</td>
<td>Ethernet Source</td>
<td>Ports 1-52: 99</td>
</tr>
<tr>
<td>4</td>
<td>IPv4 FIB</td>
<td>Ports 1-52: 100</td>
</tr>
<tr>
<td>5</td>
<td>IPv6 FIB</td>
<td>Ports 1-52: 100</td>
</tr>
<tr>
<td>6</td>
<td>VIP</td>
<td>Ports 1-52: 20</td>
</tr>
<tr>
<td>7</td>
<td>Ethernet Destination</td>
<td>Ports 1-52: 99</td>
</tr>
<tr>
<td>8</td>
<td>Flood</td>
<td>Ports 1-52: 280</td>
</tr>
</tbody>
</table>

* 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,
  ..............
  ..............

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
```
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) MUST be set to Aruba in the configuration file.

Listing 47: /etc/faucet/faucet.yaml

```yaml
dps:
aruba-3810:
    dp_id: <DP ID from *show openflow instance aggregate | include Datapath ID>
    hardware: "Aruba"
    interfaces:
        1:
            native_vlan: 100
        2:
            native_vlan: 100
```

Debug

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

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

Debug Logging

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

Enabled debug types:
openflow
openflow packets
openflow events
openflow errors
openflow packets tx
openflow packets rx
openflow packets tx pkt_in
openflow packets rx pkt_out
openflow packets rx flow_mod
```
PKI setup on switch (OPTIONAL)

Only complete this section if you wish to secure the OpenFlow connection between switch and FAUCET with certificates.

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

```plaintext
// Configure DNS. Here DNS is set to a local LAN DNS server
switch (config)# ip dns server-address priority 1 10.0.0.1

switch# show crypto pki application

Certificate Extension Validation :

<table>
<thead>
<tr>
<th>Application</th>
<th>SAN/CN</th>
</tr>
</thead>
<tbody>
<tr>
<td>openflow</td>
<td>Disabled</td>
</tr>
<tr>
<td>syslog</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

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

// Copy the root certificate for the EXAMPLE_CA from a tftp server
switch# copy tftp ta-certificate EXAMPLE_CA 10.0.0.1 myswitch.cert.pem

switch# show crypto pki ta-profile EXAMPLE_CA

<table>
<thead>
<tr>
<th>Profile Name</th>
<th>Profile Status</th>
<th>CRL Configured</th>
<th>OCSP Configured</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLE_CA</td>
<td>1 certificate installed</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

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 set up.

switch (config)# crypto pki identity-profile hpe_sf_switch1 subject common-name myswitch org MyOrgName org-unit MyOrgUnit locality MyCity state CA country US

switch (config)# show crypto pki identity-profile

Switch Identity:
ID Profile Name : hpe_sf_switch1
Common Name (CN) : myswitch
Org Unit (OU) : MyOrgUnit
Org Name (O) : MyOrgName
Locality (L) : MyCity
State (ST) : CA
Country (C) : US

// Generate CSR
switch (config)# crypto pki create-csr certificate-name hpeswt_switch1_crt ta-profile EXAMPLE_CA usage openflow
```
// Copy the printed CSR request and send it to "EXAMPLE_CA"

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

// Once the signed certificate is received, copy the same to switch.
switch (config)# copy tftp local-certificate 10.0.0.1 myswitch.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 EXAMPLE_CA

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 48: /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 49: /etc/faucet/faucet.yaml

```
vlans:
  100:
    name: "test"
dps:
  lagopus-1:
    dp_id: 0x1
    hardware: "Lagopus"
    interfaces:
      1:
        native_vlan: 100
      2:
        native_vlan: 100
```
**Start Lagopus**

Start in debug mode, in a dedicated terminal.

```
lagopus -d
```

**Run FAUCET**

```
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 50: /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:e0:87  
   →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 51: conf-zodiac.sh

```bash
#!/usr/bin/expect
#
## configure ZodiacFX with recommended settings.
#
# Serial port assigned to ZodiacFX
set port /dev/ttyACM0

# ZodiacFX network settings
set configip "10.0.1.99"
set confignetmask "255.255.255.0"
set configgateway "10.0.1.1"

# OpenFlow controller network settings
set configofcontroller "10.0.1.8"
set configofport 6653

set timeout 5
set prompt {Zodiac_FX
[185]#}
set configprompt {Zodiac_FX
[217](config)
[228]#}
set spawned [spawn -open [open $port w+]]

send_user "get initial prompt\n"
send \r
send \r
expect -re $prompt
send_user "found initial prompt\n"
send "config\r"
expect -re $configprompt
send_user "setting ethertype-filter\n"
send "set ethertype-filter enable\r"
expect -re $configprompt
send_user "setting IP address\n"
send "set ip-address $configip\r"
expect -re $configprompt
```

(continues on next page)
Example of running the script:

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

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 ZodiacGX

Introduction

ZodiacGX is a small 5 port multi table OF1.3 switch from Northbound Networks. Please see the documentation for configuring OpenFlow on the switch, and use ZodiacGX as the FAUCET hardware type.

Caveats

- The default OF port is 6633; it is recommended to use 6653.
- Minimum firmware required is v1.01

1.7.6 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 NoviWare versions starting from NW400.5.4, running with FAUCET v1.8.14.

Compared to older versions of NoviWare and Faucet, where manual pipeline configuration was required, it is possible to use the GenericTFM Hardware type to make Faucet automatically program the tables based on the needs of its current configuration.

Setup

Configure the CPN on the switch

The only configuration required in the switch is the definition of the IP and ports on which the Faucet controller must be reached. Optionally it is also possible to change the switch DPID. In this example, the server running FAUCET is 10.0.1.8; configuration for CPN interfaces is not shown.

```bash
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
```

Create faucet.yaml

In order to exploit the automatic pipeline configuration, the hardware specified in faucet.yaml must be GenericTFM

```yaml
vlans:
  100:
    name: "test"

dps:
  noviflow-1:
    dp_id: 0x1
    hardware: "GenericTFM"
```

(continues on next page)
interfaces:
  1:
    native_vlan: 100
  2:
    native_vlan: 100
  etc...

Run FAUCET

faucet --verbose

Using Older Faucet and NoviWare versions

Before the introduction of GenericTFM, Faucet used a static pipeline which needed to be configured in the switch before connecting to the controller. The following match configuration is known to pass the unit tests using NW400.4.3 with 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.

```plaintext
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 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:

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

1.7.7 Faucet on Cisco Switches

Introduction

Cisco supports Openflow with faucet pipeline on the Catalyst 9000 Series switches.

Cisco IOS XE first introduced faucet support in version 16.9.1, however since faucet support is being continually improved on Cisco platforms we recommend running the latest stable release. Currently we would recommend running 16.12.1c or later.

For official Cisco documentation on OpenFlow and faucet support see the following configuration guide:

- Programmability Configuration Guide, Cisco IOS XE Gibraltar 16.12.x
Faucet Documentation

Setup

Boot up in Openflow Mode

The Catalyst 9K will be in traditional switching mode by default. The below command will enable Openflow mode on the switch.

```
Switch-C9300#
Switch-C9300#configure terminal
Switch-C9300(config)#boot mode ?
openflow openflow forwarding mode

Switch-C9300(config)#boot mode openflow
Changes to the boot mode preferences have been stored, but it cannot take effect until the next reload.
Use "show boot mode" to check the boot mode currently active.
Switch-C9300(config)#end

Switch-C9300#show boot mode
System initialized in normal switching mode
System configured to boot in openflow forwarding mode
Reload required to boot switch in configured boot mode.
Switch-C9300#reload
```

Configure Openflow

** Configure the Management interface communicate with controller. **

```
Switch-C9300#
Switch-C9300#configure terminal
Switch-C9300(config)#interface GigabitEthernet0/0
Switch-C9300(config-if)#vrf forwarding Mgmt-vrf
Switch-C9300(config-if)#ip address 192.168.0.41 255.255.255.0
Switch-C9300(config-if)#negotiation auto
Switch-C9300(config)#end

Switch-C9300#configure terminal
Switch-C9300(config)#feature openflow
Switch-C9300(config)#openflow
Switch-C9300(config-openflow)#switch 1 pipeline 1
Switch-C9300(config-openflow-switch)#controller ipv4 192.168.0.91 port 6653 vrf Mgmt-vrf security none
Switch-C9300(config-openflow-switch)#controller ipv4 192.168.0.91 port 6654 vrf Mgmt-vrf security none
Switch-C9300(config-openflow-switch)#datapath-id 0xABCDEF1234
Switch-C9300(config-openflow-switch)#end
```

** Disable DTP/keepalive on OpenFlow ports which may interfere with FAUCET. **

```
Switch-C9300#
```
The following example will disable DTP and keepalives for TenGigabitEthernet1/0/1-24; adjust the range as necessary.

```
Switch-C9300(config)#interface range TenGigabitEthernet1/0/1-24
Switch-C9300(config-if-range)#switchport mode trunk
Switch-C9300(config-if-range)#switchport nonegotiate
Switch-C9300(config-if-range)#spanning-tree bpdufilter enable
Switch-C9300(config-if-range)#no keepalive
Switch-C9300(config-if-range)#exit
```

**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 CiscoC9K in the configuration file.

```
:dps:
  Cisco-C9K:
    dp_id: 0xABCDEF1234
    hardware: "CiscoC9K"
    interfaces:
      1:
        native_vlan: 100
        name: "port1"
      2:
        native_vlan: 100
        name: "port2"
```

**Troubleshooting**

Command to check overall openflow configuration

```
Switch-C9300#
Switch-C9300#show openflow switch 1
Logical Switch Context
  Id: 1
  Switch type: Forwarding
  Pipeline id: 1
  Data plane: secure
  Table-Miss default: drop
  Configured protocol version: Negotiate
  Config state: no-shutdown
  Working state: enabled
  Rate limit (packet per second): 0
  Burst limit: 0
  Max backoff (sec): 8
  Probe interval (sec): 5
  TLS local trustpoint name: not configured
  TLS remote trustpoint name: not configured
  Logging flow changes: Disabled
  Stats collect interval (sec): 5
  Stats collect Max flows: 9216
```
Stats collect period (sec): 1
Minimum flow idle timeout (sec): 10
OFA Description:
Manufacturer: Cisco Systems, Inc.
Hardware: C9300-48P
Software: Cisco IOS Software (Fiji), Catalyst L3 Switch Software (CAT9K-IOSXE), Version 16.8.1GO3, RELEASE SOFTWARE (fc1)| openvswitch 2.1
Serial Num: FCW2145L0FP
DP Description: Faucet-C9300:sw1
OF Features:
DPID: 0x000000ABCDEF1234
Number of tables: 9
Number of buffers: 256
Capabilities: FLOW_STATS TABLE_STATS PORT_STATS
Controllers:
192.168.0.91:6653, Protocol: TCP, VRF: Mgmt-vrf
192.168.0.91:6654, Protocol: TCP, VRF: Mgmt-vrf
Interfaces:
GigabitEthernet1/0/1
GigabitEthernet1/0/2

Command to check the openflow flows installed

Switch-C9300#
Switch-C9300#show openflow switch 1 flow list
Logical Switch Id: 1
Total flows: 9

Flow: 1 Match: any Actions: drop, Priority: 0, Table: 0, Cookie: 0x0, Duration: 33812.029s, Packets: 46853, Bytes: 3636857

Command to check the state of the port status

Switch-C9300#
Switch-C9300#show openflow switch 1 ports
Logical Switch Id: 1

<table>
<thead>
<tr>
<th>Port</th>
<th>Interface Name</th>
<th>Config-State</th>
<th>Link-State</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gi1/0/1</td>
<td>PORT_UP</td>
<td>LINK_UP</td>
<td>1GB-HD</td>
</tr>
<tr>
<td>2</td>
<td>Gi1/0/2</td>
<td>PORT_UP</td>
<td>LINK_DOWN</td>
<td>1GB-HD</td>
</tr>
<tr>
<td>3</td>
<td>Gi1/0/3</td>
<td>PORT_UP</td>
<td>LINK_DOWN</td>
<td>1GB-HD</td>
</tr>
<tr>
<td>4</td>
<td>Gi1/0/4</td>
<td>PORT_UP</td>
<td>LINK_DOWN</td>
<td>1GB-HD</td>
</tr>
</tbody>
</table>

Command to check the status of the controller

Switch-C9300#
Switch-C9300#show openflow switch 1 controller
Logical Switch Id: 1
Total Controllers: 2

Controller: 1
192.168.0.91:6653
Protocol: tcp
VRF: Mgmt-vrf
Connected: Yes
Role: Equal
Negotiated Protocol Version: OpenFlow 1.3
Last Alive Ping: 2018-10-03 18:43:07 NZST
state: ACTIVE
sec_since_connect: 13150

Controller: 2
192.16.0.91:6654
Protocol: tcp
VRF: Mgmt-vrf
Connected: Yes
Role: Equal
Negotiated Protocol Version: OpenFlow 1.3
Last Alive Ping: 2018-10-03 18:43:07 NZST
state: ACTIVE
sec_since_connect: 12960

Command to check controller statistics

Switch-C9300#
Switch-C9300#show openflow switch 1 controller stats
Logical Switch Id: 1
Total Controllers: 2

Controller: 1
address : tcp:192.168.0.91:6653%Mgmt-vrf
connection attempts : 165
successful connection attempts : 61
flow adds : 1286700
flow mods : 645
flow deletes : 909564
flow removals : 0
flow errors : 45499
flow unencodable errors : 0
total errors : 45499
echo requests : rx: 842945, tx:205
echo reply : rx: 140, tx:842945
flow stats : rx: 0, tx:0
barrier : rx: 8324752, tx:8324737
packet-in/packet-out : rx: 29931732, tx:8772758

Controller: 2
address : tcp:192.168.0.91:6654%Mgmt-vrf
connection attempts : 11004
successful connection attempts : 3668
flow adds : 0
flow mods : 0
flow deletes : 0
flow removals : 0
flow errors : 0
flow unencodable errors : 0
total errors : 0
echo requests : rx: 946257, tx:1420
echo reply : rx: 1420, tx:946257
flow stats : rx: 47330, tx:57870
barrier : rx: 0, tx:0
packet-in/packet-out : rx: 2377, tx:0
1.7.8 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:

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

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

```bash
$ ./tools/dpdk-devbind.py --status
```

<table>
<thead>
<tr>
<th>Network devices using DPDK-compatible driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network devices using kernel driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:01:00.0 '82580 Gigabit Network Connection' if=enp1s0f0 drv=igb unused=</td>
</tr>
<tr>
<td>0000:01:00.1 '82580 Gigabit Network Connection' if=enp1s0f1 drv=igb unused=</td>
</tr>
<tr>
<td>0000:01:00.2 '82580 Gigabit Network Connection' if=enp1s0f2 drv=igb unused=</td>
</tr>
<tr>
<td>0000:01:00.3 '82580 Gigabit Network Connection' if=enp1s0f3 drv=igb unused=</td>
</tr>
</tbody>
</table>

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

Configure an OVS bridge with the DPDK ports

ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev protocols=OpenFlow13
ovs-vsctl add-port br0 dpdk0 -- set interface enp1s0f0 type=dpdk options:dpdk-devargs=0000:01:00.0
ovs-vsctl add-port br0 dpdk1 -- set interface enp1s0f1 type=dpdk options:dpdk-devargs=0000:01:00.1
ovs-vsctl set-fail-mode br0 secure
ovs-vsctl set-controller br0 tcp:127.0.0.1:6653
ovs-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 52: /etc/faucet/faucet.yaml

```
vans:
  100:
    name: "test"

dps:
  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 53: /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) 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) 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
```
1.7.9 Faucet Testing with OVS on Hardware

Setup

![Network Diagram]  

Faucet configuration file

```yaml
Listing 54: /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
```

1.7. Vendor-specific Documentation
Faucet Documentation

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)

```bash
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
   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
   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
   inet6 fe80::b696:91ff:fe00:88a5/64 scope link
     valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default
   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
   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
   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
   link/eno1/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
   inet6 fe80::b696:91ff:fe00:88a8/64 scope link
     valid_lft forever preferred_lft forever
   inet6 fe80::b696:91ff:fe00:88a9/64 scope link
     valid_lft forever preferred_lft forever
```

(continues on next page)
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 OpenSwitch and start `openvswitch-switch service`

```bash
$ 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"
```

```bash
$ sudo ovs-vsctl -- --columns=name,ofport list Interface
  name : "ovs-br0"
    ofport : 65534
  name : "enp5s0"
    ofport : 3
  name : "enp2s0"
    ofport : 1
```
<table>
<thead>
<tr>
<th>name</th>
<th>&quot;enp6s0&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofport</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>&quot;enp3s0&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofport</td>
<td>2</td>
</tr>
</tbody>
</table>

**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

```bash
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 10MB-FD 100MB-HD 100MB-FD 1GB-HD 1GB-FD COPPER AUTO_NEG AUTO_PAUSE AUTO_PAUSE_ASYM
  supported: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 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.

```bash
$ 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

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

Or

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

To read the pcap file, use

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

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

```bash
$ 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.

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


1.8 External Resources

1.8.1 Online Tutorials


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

- If you have general questions, feel free to reach out to the faucet-dev mailing list.
- If you are new to FAUCET, or are contemplating a major change, it’s recommended to open a github issue with the proposed change. This will enable broad understanding of your work including being able to catch any potential snags very early (for example, adding new dependencies). Architectural and approach questions are best settled at this stage before any code is written.
- Please send relatively small, tightly scoped PRs (approx 200-300 LOC or less). This makes review and analysis easier and lowers risk, including risk of merge conflicts with other PRs. Larger changes must be refactored into incremental changes.
- You must add a test if FAUCET’s functionality changes (ie. a new feature, or correcting a bug).
- All unit and integration tests must pass (please use the docker based tests; see Software switch testing with docker). Where hardware is available, please also run the hardware based integration tests also.
- In order to speed up acceptance of your PR we recommend enabling TravisCI on your own github repo, and linking the test results in the body of the PR. This enables the maintainers to quickly verify that your changes pass all tests in a pristine environment while conserving our TravisCI resources on the main branch (by minimizing resources used on potentially failing test runs which could be caught before opening a PR on the main branch).
- You must use the github feature branches (see https://gist.github.com/vlandham/3b2b79c40bc7353ae95a), for your change and squash commits (https://blog.github.com/2016-04-01-squash-your-commits/) when creating the PR.
- 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, which will save you TravisCI resources also.
- pylint must show no new errors or warnings.
- Code must conform to the style guide (see below).
2.1.2 PR handling guidelines

This section documents general guidelines for the maintainers in handling PRs. The overall intent is, to enable quality contributions with as low overhead as possible, maximizing the use of tools such as static analysis and unit/integration testing, and supporting rapid and safe advancement of the overall project.

In addition to the above PR submission guidelines, above:

- PRs require a positive review per github’s built in gating feature. The approving reviewer executes the merge.
- PRs that should not be merged until some other criteria are met (e.g. not until release day) must include DO NOT MERGE in the title, with the details in PR comments.
- A typical PR review/adjust/merge cycle should be 2-3 days (timezones, weekends, etc permitting). If a PR upon review appears too complex or requires further discussion it is recommended it be refactored into smaller PRs or discussed in another higher bandwidth forum (e.g. a VC) as appropriate.
- A PR can be submitted at any time, but to simplify release logistics PR merges might not be done before release, on release days.

2.1.3 Code style

Please use the coding style documented at https://github.com/google/styleguide/blob/gh-pages/pyguide.md. Existing code not using this style will be incrementally migrated to comply with it. New code should comply.

2.1.4 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 below.

If you would rather develop on the command line directly, a short summary of the command line setup for development in a virtualenv with Python 3.6+ is included after the PyCharm instructions.

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 if we open one of the code files for faucet (e.g. faucet/faucet.py) 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.

If you are using the sample configuration “as is" you will also need to copy /Dev/faucet/etc/faucet/acls. yaml to /Dev/faucet/venv/etc/faucet/ as that included by the sample faucet.yaml file, and without it the sample faucet.yaml file cannot be loaded.

You may also wish to copy /Dev/faucet/etc/faucet/ryu.conf to /Dev/faucet/venv/etc/faucet/ as well so everything can be referenced in one directory inside the Python virtual environment.

**Configure PyCharm to run faucet and gauge**

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 now add run configuration for starting faucet and gauge. Click the + button in the top left hand corner of the window. First, change the name from Unnamed to faucet. 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 use the same steps as above to add a run configuration for gauge. Changing the Script path to ..../ venv/bin/ryu-manager and setting the Parameters this time to faucet.gauge. Make sure the working directory is set to /Dev/faucet/faucet/.
Configure PyCharm to run unit tests

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
```

To add the test run configuration we will again click the + button in the top left hand corner, 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/unit/faucet. For Pattern enter test_*.py.

We will also add test run configuration for gauge using the same steps as above. Use Gauge Unit Tests as the Name and for Target select Script path and enter the path /Dev/faucet/tests/unit/gauge. 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.

Developing with a Python 3.6+ venv

If you would prefer not to use PyCharm and are comfortable developing Python directly on the command line, these steps should get you started. They have been tested with Ubuntu 18.04 LTS, which includes Python 3.6, but similar instructions should work on other platforms that include Python 3.6+.

Install C/C++ compilers and Python development environment packages:

```
sudo apt-get install python3-venv libpython3.6-dev gcc g++ make
```

If you have not already, clone the faucet git repository:

```
git clone https://github.com/faucetsdn/faucet.git
```

Then create a Python venv environment within it:

```
   cd faucet
   python3 -m venv "$PWD/venv"
```

and activate that virtual environment for all following steps:

```
   . venv/bin/activate
```

Ensure that the faucet config is present within the virtual environment, copying from the default config files if required:

```
   mkdir -p "$VIRTUAL_ENV/var/log/faucet"
   mkdir -p "$VIRTUAL_ENV/etc/faucet"

   for FILE in {acls,faucet,gauge}.yaml ryu.conf; do
      if [ -f "$VIRTUAL_ENV/etc/faucet/$(FILE)" ]; then
         echo "Preserving existing $(FILE)"
      else
         echo "Installing template $(FILE)"
         cp -p "etc/faucet/$(FILE)" "$VIRTUAL_ENV/etc/faucet/$(FILE)"
      fi
   done
```

Then install the runtime and development requirements.
Finally install faucet in an editable form:

```
pip install -e .
```

And then confirm that you can run the unit tests:

```
pytest tests/unit/faucet/
ppytest tests/unit/gauge/
```

### 2.1.5 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.

### Building Documentation

The documentation is built with Sphinx, from within the `docs` directory.

To be able to build the documentation ensure you have the relevant packages installed:

```
cd docs
sudo apt-get install librsvg2-bin make
pip3 install -r requirements.txt
```

and then you can build HTML documentation with:

```
cd docs
make html
```

and the documentation will be found under `_build/html` in the `docs` directory.
2.1.6 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.

130 Chapter 2. Developer Documentation
2.2.2 Faucet Openflow Switch Pipeline

This summarizes the global FAUCET pipeline; however, certain tables may be omitted if the functionality is not required. For example, if routing is not configured, neither FIB table nor the VIP table will be provisioned.

Usually the OpenFlow table IDs will be allocated sequentially for the tables actually used, so tables should be referenced by their name rather than the table ID in this diagram.

See also canonical pipeline definitions in faucet_pipeline.py.

PORT_ACL Table

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

VLAN Table

- 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
Faucet Documentation

* Push VLAN frame onto packet with VLAN_VID representing ports native VLAN and send to next table
  - Interception of L2 control traffic (e.g. LACP, LLDP if configured).
  - Unknown traffic is dropped

**VLAN_ACL Table**

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

**ETH_SRC Table**

- **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

**IPV4_FIB Table**

- **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/HAIRPIN/VIP table
  - Unknown traffic is dropped

**IPV6_FIB Table**

- **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/HAIRPIN/VIP table
  - Unknown traffic is dropped
VIP Table

- Match fields: \(\text{arp}_\text{tpa}, \text{eth}_\text{dst}, \text{eth}_\text{type}, \text{icmpv6}_\text{type}, \text{ip}_\text{proto}\)
- Operations:
  - Send traffic destined for FAUCET VIPs including IPv4 ARP and IPv6 ND to the controller, and traffic for unresolved hosts in connected IP subnets (if proactively learning).
  - IPv4 ARP/IPv6 ND traffic may be flooded also (sent to FLOOD)

ETH_DST_HAIRPIN Table

- Exact match (no wildcards)
- Match fields: \(\text{eth}_\text{dst}, \text{in}_\text{port}, \text{vlan}_\text{vid}\)
- Operations:
  - For destination MAC addresses we have learned output packet towards that host (popping VLAN frame if we are outputting on an untagged port), and where hairpinning is desired (e.g. routing between hosts on the same port, but different VLANS).
  - Unknown traffic is sent to ETH_DST table.

ETH_DST Table

- Exact match (no wildcards)
- Match fields: \(\text{eth}_\text{dst}, \text{vlan}_\text{vid}\)
- 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

FLOOD Table

- Match fields: \(\text{eth}_\text{dst}, \text{in}_\text{port}, \text{vlan}_\text{vid}\)
- 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

You can build and run the mininet tests with the following commands:

```
sudo docker build --pull -t faucet/tests -f Dockerfile.tests .
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.tcpdump
sudo modprobe openvswitch
sudo docker run --name=faucet-tests \
   --sysctl net.ipv6.conf.all.disable_ipv6=0 --privileged --rm \
   -v /var/local/lib/docker:/var/lib/docker \
   -v /tmp/faucet-pip-cache:/var/tmp/pip-cache \
   -ti faucet/tests
```
The apparmor command is currently required on Ubuntu hosts to allow the use of tcpdump inside the container.

If you need to use a proxy, the following to your docker run command.

```
--build-arg http_proxy=http://your.proxy:port
```

### 2.3.3 Hardware switch testing with docker

```
  +--------------------------+
  |                         |
  |  FAUCET CPN              |
  |                         |
  +--------------------------+
  |                         |
  |  FAUCET test host        |
  |                         |
  |  +--+                   +--
  |  |                     | |
  |  |                     | |
  |  FAUCET test host        |
  |  +--+                   +--
  |                     |
  |                     |
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
  |                     |
  |                     |
  |                     |
  +---------------------+
```
Faucet Documentation

Requirements

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

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

---

**Note:** It is very important to disable any process that could cause any traffic on the dataplane test interfaces, and 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 the test interfaces.

---

**Note:** Hardware tests must not be run from virtualized hosts (such as under VMware). The tests need to control physical port status, and need low level L2 packet access (eg. to rewrite Ethernet source and destination addresses) which virtualization may interfere with.

---

**Note:** Hardware tests require the test switch to have all non-OpenFlow switching/other features (eg. RSTP, DHCP) disabled on the dataplane test interfaces. These features will conflict with the functions FAUCET itself provides (and in turn the tests).

---

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:

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

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

```yaml
hw_switch: True
hardware: 'Open vSwitch'
# Map ports on the hardware switch, to physical ports on this machine.
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
```

(continues on next page)
**Running the tests**

Before starting the hardware test suite for the first time, you will need to install ebtables on the host machine:

```bash
sudo apt-get install ebtables
```

After every reboot of your host machine you will also need to manually load the `openvswitch` and `ebtables` kernel modules. If using apparmor you will also need to disable the profile for `tcpdump`:

```bash
sudo modprobe openvswitch
sudo modprobe ebtables
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.tcpdump
```

Then you can build and run the test suite:

```bash
sudo docker build --pull -t faucet/tests -f Dockerfile.tests .
sudo docker run --name=faucet-tests \
   --privileged --rm --net=host --cap-add=NET_ADMIN \
   -v /var/local/lib/docker:/var/lib/docker \
   -v /tmp/faucet-pip-cache:/var/tmp/pip-cache \
   -v /etc/faucet:/etc/faucet \
   -v /var/tmp:/var/tmp \
   -ti faucet/tests
```

### 2.3.4 Test suite options

In both the software and hardware version of the test suite we can provide flags inside the `FAUCET_TESTS` environment variable to run specific parts of the test suite.

**Note:** Multiple flags can be added to `FAUCET_TESTS`, below are just some examples of how individual flags work.

To find the full list of options you can pass to the test suite, set `FAUCET_TESTS` to `--help`.

```bash
-e FAUCET_TESTS="--help"
```
Running specific integration tests

If specific test names are listed in the `FAUCET_TESTS` environment then only these integration tests will be run and all others skipped.

If we add the following to either of the previous docker run commands then only the `FaucetUntaggedTest` will be run.

```
-e FAUCET_TESTS="FaucetUntaggedTest"
```

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.

```
-e FAUCET_TESTS="-i"
```

Skip code checks

Sometimes you will want to skip the pytype, linting and documentation tests. This can be done with with the `-n` flag:

```
-e FAUCET_TESTS="-n"
```

Skip unit tests

Sometimes you will want to skip the unit tests which are small tests that verify small chunks of the code base return the correct values. If these are skipped the integration tests (which spin up virtual networks and tests faucet controllers under different configurations) will still be run.

This can be done with with the `-u` flag:

```
-e FAUCET_TESTS="-u"
```

Checking test results

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

By default the test suite cleans up these files but if we use the `-k` flag the test suite will keep these files.

```
-e FAUCET_TESTS="-k"
```
2.3.5 Repeatedly running tests until failure

You can run tests until a failure is detected (eg, to diagnose an unreliable test). Tests will continue to run forever until at least one fails or the test is interrupted.

```bash
-e FAUCET_TESTS="-r"
```

2.3.6 Test debugging

Often while debugging a failed integration test it can be useful to pause the test suite at the point of the failure. The test can then be inspected live to narrow down the exact issue. To do this, run your test with the `--debug` flag (replace `TEST_NAME` with actual name of test).

```bash
-e FAUCET_TESTS="--debug TEST_NAME"
```

The test suite will now run in a mode where it ignores successful tests and drops into a pdb shell when a failure occurs inside a test. There are a number of different pdb commands that can be run to check the actual test code.

It is also possible to login to the virtual container environment to run interactive debug commands to inspect the state of the system.

```bash
sudo docker exec -it faucet-tests /bin/bash
```

One useful thing can be to find the running mininet containers and execute commands inside of them, e.g ping:

```
root@35b98943f736:/faucet-src# ps w | grep mininet:
   995 pts/1  Ss+   0:00 bash --norc --noediting -is mininet:faucet-637
   997 pts/2  Ss+   0:00 bash --norc --noediting -is mininet:u021
  1001 pts/3  Ss+   0:00 bash --norc --noediting -is mininet:u022
  1005 pts/4  Ss+   0:00 bash --norc --noediting -is mininet:u023
  1009 pts/5  Ss+   0:00 bash --norc --noediting -is mininet:u024
  1013 pts/6  Ss+   0:00 bash --norc --noediting -is mininet:s02
  1077 pts/7  Ss+   0:00 bash --norc --noediting -is mininet:gauge-637

root@35b98943f736:/faucet-src# m u021 ping 127.0.0.1
```

2.4 Fuzzing

2.4.1 Fuzzing faucet config with docker

First, get yourself setup with docker based on our Installing 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
```

2.4. Fuzzing
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:

```bash
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:

```bash
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.

```python
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.
```

140 Chapter 2. Developer Documentation
• 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:
• tunnel (dict): the tunnel formation, creates a tunnel from the applied port(s) to the specified destination
• 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
• 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': <class 'str'>, 'mirror': (<class 'str'>, <class 'int'>), 'output': <class 'dict'>}

build(meters, vid, port_num)
Check that ACL can be built from config.

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

defaults = {'dot1x_assigned': False, 'exact_match': False, 'rules': None}
defaults_types = {'dot1x_assigned': <class 'bool'>, 'exact_match': <class 'bool'>, 'rules': <class 'list'>}

finalize()
Configuration parsing marked complete.

get_in_port_match(tunnel_id)
Returns a port number of the src_port of the tunnel that the ingress tunnel ACL will need to match to.
:param tunnel_id: tunnel identifier to obtain the src_port :type tunnel_id: int

Returns src_port number if it exists, None otherwise
Return type int OR None

get_meters()
get_mirror_destinations()
get_tunnel_id(rule_index)
Gets the tunnel ID for the rule :param_rule_index: Index of the tunnel rule in the self.rules list :type rule_index: int

Returns Identifier for the tunnel
Return type tunnel_id (int)

get_tunnel_rule_indices()
Get the rules from the rule conf that contain tunnel outputs :returns: list of integer indices into the self.rules
rule list that contain tunnel information :rtype: rules (list)
output_actions_types = {'failover': <class 'dict'>, 'pop_vlans': <class 'int'>, 'port': (<class 'str'>, <class 'int'>), ... 'list'>, 'swap_vid': <class 'int'>, 'tunnel': <class 'dict'>, 'vlan_vid': <class 'int'>, 'vlan_vids': <class 'list'>}

remove_non_tunnel_rules()
Removes all non-tunnel rules from the ACL and removes all match fields and non-tunnel required actions from the tunnel rules

resolve_ports (resolve_port_cb, resolve_tunnel_objects)

rule_types = {'actions': <class 'dict'>, 'arp_op': (<class 'str'>, <class 'int'>), 'arp_sha': (<class 'str'>, <class 'int'>), ...

tunnel_types = {'dp': <class 'str'>, 'port': (<class 'str'>, <class 'int'>), 'tunnel_id': (<class 'str'>, <class 'int'>), 'type': <class 'str'>}

unpack_tunnel (tunnel_id)
Retrieves the information from the tunnel dict for the tunnel with id :param tunnel_id: Identifier for the tunnel :type tunnel_id: int

Returns Tunnel information

Return type (src_dp, src_port, dst_dp, dst_port)

update_tunnel_acl_conf (dp)
Update the ACL rule conf if the DP is in the path :param dp: The dp that this tunnel acl object belongs to :type dp: DP

Returns True if any value was updated

Return type bool

verify_tunnel_rules (dp)
Verify the actions in the tunnel ACL to by making sure the user hasn’t specified an action/match that will create a clash. :param dp: The dp that this tunnel acl object belongs to :type dp: DP

TODO: Choose what combinations of matches & actions to disallow with a tunnel

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 ()
Mainline.

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_diff (other)
Return text diff between two Confs.

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

mutable_attrs = frozenset({})

set_defaults(defaults=None, conf=None)
    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)
    Evaluate condition and raise InvalidConfigError if condition not True.

faucet.config_parser module

Implement configuration file parsing.

faucet.config_parser.dp_parser(config_file, logname, meta_dp_state=None)
    Parse a config file into DP configuration objects with hashes of config include/files.

faucet.config_parser.dp_preparsed_parser(top_confs, meta_dp_state)
    Parse a preparsed (after include files have been applied) FAUCET config.

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_contents, 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.

        DEFAULT_LLDP_MAX_PER_INTERVAL = 5

        DEFAULT_LLDP_SEND_INTERVAL = 5

        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.
**Faucet Documentation**

```python
classmethod add_stack_link(graph, dp, port)
    Add a stack link to the stack graph.

all_lags_up()
    Return True if all LAGs have at least one port up.

any_stack_port_up()
    Return True if any stack port is up.

base_prom_labels()
    Return base Prometheus labels for this DP.

bgp_routers()
    Return list of routers with BGP enabled.

static canonical_port_order(ports)

check_config()
    Check configuration of this dp

classification_table()
    Returns classification table

clone_dyn_state(prev_dp)
    Clone dynamic state for this dp

cold_start(now)
    Update to reflect a cold start

coprocessor_ports()
    Return list of coprocessor ports.

default_table_sizes_types = {'classification': <class 'int'>, 'eth_dst': <class 'int'>, ... <class 'int'>, 'port_acl': <class 'int'>, 'vip': <class 'int'>, 'vlan': <class 'int'>, 'vlan_acl': <class 'int'>}
defaults = {'advertise_interval': 30, 'arp_neighbor_timeout': 30, 'cache_update_guard_time': 0, ... True, 'table_sizes': {}, 'timeout': 300, 'use_classification': False, 'use_idle_timeout': False}
defaults_types = {'advertise_interval': <class 'int'>, 'arp_neighbor_timeout': <class 'int'>, 'cache_update_guard_time': <class 'int'>}
dot1x_defaults_types = {'auth_acl': <class 'str'>, 'nfv_intf': <class 'str'>, 'nfv_sw_port': <class 'int'>, 'noauth_acl': <class 'str'>, 'radius_ip': <class 'str'>, 'radius_port': <class 'int'>, 'radius_secret': <class 'str'>}
dot1x_ports()
    Return list of ports with 802.1x enabled.

finalize()
    Need to configure OF tables as very last step.

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

finalize_tunnel_acls(dps)
    Turn off ACLs not in use and resolve the ACL src dp and port.
    Parameters
dps (list) – DPs.

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. deleted_meters (set): deleted meter numbers changed_meters (set): changed/added meter numbers

Return type (tuple)

def get_config_dict():
    "Return DP config as a dict for API call."
def get_native_vlan(port_num):
    "Return native VLAN for a port by number, or None."
def get_node_link_data():
    "Return network stacking graph as a node link representation"
def get_tables():
    "Return tables as dict for API call."
def in_port_tables():
    "Return list of tables that specify in_port as a match."

def is_in_path(src_dp, dst_dp):
    "Return True if the current DP is in the path from src_dp to dst_dp"

    Parameters
    • src_dp (DP) – DP
    • dst_dp (DP) – DP

    Returns True if self is in the path from the src_dp to the dst_dp.

    Return type bool

def is_stack_edge():
    "Return True if this DP is a stack edge."

def is_stack_root():
    "Return True if this DP is the root of the stack."

def is_stack_root_candidate():
    "Return True if this DP could be a root of the stack."

def is_transit_stack_switch():
    "Return true if this is a stack switch in reset_refs self.stack might not be configured yet"

def lacp_ports():
    "Return ports that have LACP."

def lacp_up_ports():
    "Return ports that have LACP up."

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

def lags_up():
    "Return dict of LAGs mapped to member ports that have LACP up."

def lldp_beacon_defaults_types = {'max_per_interval': <class 'int'>, 'send_interval': <class 'int'>, 'system_name': <class 'str'>}

def lldp_beacon_send_ports(now):
    "Return list of ports to send LLDP packets; stacked ports always send LLDP."
match_tables (match_type)
    Return list of tables with matches of a specific match type.

static modify_stack_topology (graph, dp, port, add=True)
    Add/remove an edge to the stack graph which originates from this dp and port.

mutable_attr = frozenset({'stack', 'vlans'})

non_vlan_ports ()
    Ports that don’t have VLANs on them.

output_table ()
    Returns first output table

output_tables ()
    Return tables that cause a packet to be forwarded.

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

port_labels (port_no)
    Return port name and description labels for a port number.

port_no_valid (port_no)
    Return True if supplied port number valid on this datapath.

classmethod remove_stack_link (graph, dp, port)
    Remove a stack link to the stack graph.

reset.refs (vlans=None)
    Resets VLAN references.

resolve_port (port_name)
    Resolve a port by number or name.

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

restricted_bcast_arpnd_ports ()
    Return ports that have restricted broadcast set.

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

shortest_path (dest_dp, src_dp=None)
    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 (src_dp=None)
    Return shortest path to root DP, as list of DPs.

stack_defaults_types = {'priority': <class 'int'>}

stack_longest_path_to_root_len ()
    Return length of the longest path to root in the stack.

table_by_id (table_id)
    Gets first table with table id
Faucet Documentation

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.EventFaucetEventSockHeartbeat
    Bases: ryu.controller.event.EventBase

    Event used to trigger periodic events on event sock, causing it to raise an exception if conn is broken.

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

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

class faucet.faucet.EventFaucetFastAdvertise
    Bases: ryu.controller.event.EventBase

    Event used to trigger periodic fast network advertisements (eg LACP).

class faucet.faucet.EventFaucetFastStateExpire
    Bases: ryu.controller.event.EventBase

    Event used to trigger fast expiration of state in controller.

class faucet.faucet.EventFaucetMaintainStackRoot
    Bases: ryu.controller.event.EventBase

    Event used to maintain stack root.

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.

event_socket_heartbeat_time = 0

exc_logname = 'faucet.exception'

features_handler (ryu_event)
    Handle receiving a switch features message from a datapath.
    Parameters ryu_event (ryu.controller.ofp_event.EventOFPSwitchFeatures) – trigger.

flow_removed_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.

notifier = None

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

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.BgpSpeakerKey(dp_id, vlan_vid, ipv)
    Bases: object
    
    Uniquely describe a BGP speaker.

class faucet.faucet_bgp.FaucetBgp(logger, exc_logname, metrics, send_flow_msgs)
    Bases: object
    
    Wrapper for Ryu BGP speaker.
    
    exc_logname = None

    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_dot1x module

802.1x implementation for FAUCET.

class faucet.faucet_dot1x.FaucetDot1x(logger, exc_logname, metrics, send_flow_msgs)
    Bases: object
    
    Wrapper for experimental Chewie 802.1x authenticator.

    auth_handler(address, port_id, *args, **kwargs)
    
    Callback for when a successful auth happens.

    create_flow_pair(dp_id, dot1x_port, nfv_sw_port, valve)
    
    Creates the pair of flows that redirects the eapol packets to/from the supplicant and nfv port

    Parameters
        • dp_id(int) -
        • dot1x_port(Port) -
        • nfv_sw_port(Port) -
        • valve(Valve) -

    Returns list

    create_mab_flow(dp_id, dot1x_port, nfv_sw_port, valve)
    
    Creates a flow that mirrors UDP packets from port 68 (DHCP) from the supplicant to the nfv port

    Parameters
        • dp_id(int) -
        • dot1x_port(Port) -
        • nfv_sw_port(Port) -
        • valve(Valve) -
Returns list

def exc_logname = None

def failure_handler (address, port_id)
    Callback when an EAP failure happens.

def log_auth_event (valve, port_num, mac_str, status)
    Log an authentication attempt event

def log_port_event (event_type, port_type, valve, port_num)
    Log a dot1x port event

def logoff_handler (address, port_id)
    Callback when an EAP logoff happens.

def nfv_sw_port_up (dp_id, dot1x_ports, nfv_sw_port)
    Setup the dot1x forward port acls when the nfv_sw_port comes up.
    :param dp_id: int :param dot1x_ports: Iterable of Port objects :param nfv_sw_port: Port
    Returns list of flowmods

def port_down (dp_id, dot1x_port, nfv_sw_port)
    Remove the acls added by FaucetDot1x.get_port_acls
    :param dp_id: int :param dot1x_port: Port :param nfv_sw_port: Port
    Returns list of flowmods

def port_up (dp_id, dot1x_port, nfv_sw_port)
    Setup the dot1x forward port acls.
    :param dp_id: int :param dot1x_port: Port :param nfv_sw_port: Port
    Returns list of flowmods

def reset (valves)
    Set up a dot1x speaker.

def set_mac_str (valve, valve_index, port_num)

    Parameters
        • valve (Valve) –
        • valve_index (int) –
        • port_num (int) –

    Returns str

faucet.faucet_dot1x.get_mac_str (valve_index, port_num)
    Gets the mac address string for the valve/port combo
    :param valve_index: The internally used id of the valve.
    :type valve_index: int :param port_num: port number :type port_num: int
    Returns str
faucet.faucet_event module

FAUCET event notification.

```python
class faucet.faucet_event.FaucetEventNotifier(socket_path, metrics, logger):
    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.
```

```python
class faucet.faucet_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_experimental_api module

Implement experimental API.

```python
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.
```
**Faucet Documentation**

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

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

**faucet.faucet_metadata module**

This module contains code relating to the use of OpenFlow Metadata within Faucet.

`faucet.faucet_metadata.get_egress_metadata(port_num, vid)`
Return the metadata value to output a packet to port port_num on vlan vid

**faucet.faucet_metrics module**

Implement Prometheus statistics.

```python
def faucet.faucet_metrics.FaucetMetrics(reg=None)
    ...Container class for objects that can be exported to Prometheus.

    inc_var(var, labels, val=1)
    Reset counter for a given var/label combination.

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

**faucet.faucet_pipeline module**

Standard FAUCET pipeline.

```python
def faucet.faucet_pipeline.ValveTableConfig(name, table_id, exact_match=None, meter=None, output=True, miss_goto=None, size=None, match_types=None, set_fields=None, dec_ttl=None, vlan_scale=None, vlan_port_scale=None, next_tables=None, metadata_match=0, metadata_write=0)
    ...Configuration for a single table.
```

**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.get_samples(endpoints, metric_name, label_matches, nonzero_only=False, retries=3)`
return a list of Prometheus samples for a given metric

Prometheus Sample objects are named tuples with the fields: name, labels, value, timestamp, exemplar.

**Parameters**
Faucet Documentation

- **endpoints** (*list of strings*) – the prometheus endpoints to query
- **metric_name** (*string*) – the metric to retrieve
- **label_matches** (*dict*) – filters results by label
- **nonzero_only** (*bool*) – only return samples with non-zero values
- **retries** (*int*) – number of retries when querying

**Returns** list of Prometheus Sample objects

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='\n', 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.

```python
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'

    reload_config(ryu_event)
    Handle request for Gauge config reload.

    updateWatcher_handler(ryu_event)
    Handle any kind of stats/change event.
    Parameters ryu_event (ryu.controller.event.EventReplyBase) – stats/change event.
```

Chapter 2. Developer Documentation
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

```bash
> use faucet
Using database faucet
> show series where table_id = '0' and in_port = '2'
key
---
flow_byte_count, dp_name=windscale-faucet-1, eth_type=2048, in_port=2, ip_proto=17,
    priority=9099, table_id=0, udp_dst=53
flow_byte_count, dp_name=windscale-faucet-1, eth_type=2048, in_port=2, ip_proto=6,
    priority=9098, table_id=0, tcp_dst=53
flow_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
> 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
----  ------- ------- ------- ------- -------- ---------- ------ ------- ------- ----- ---
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
```

class faucet.gauge_influx.GaugePortStateInfluxDBLogger (conf, logname, prom_client)
    Bases: faucet.gauge_pollers.GaugePortStatePoller, faucet.gauge_influx.InfluxShipper

2.5. Source Code 155
Example

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

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

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

def 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.

    send_req ()
        Send a stats request to a datapath.

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

    Poll for all meter stats.

    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.

    static 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, 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
        • 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.

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

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.

class faucet.gauge_prom.GaugeMeterStatsPrometheusPoller (conf, logname, prom_client)
   Bases: faucet.gauge_pollers.GaugePortStatsPoller

   Exports meter stats to Prometheus.

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

   Export port state changes to Prometheus.

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

   Exports port stats to Prometheus.

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

   Wrapper for Prometheus client that is shared between all pollers.

   reregister_flow_vars (table_name, table_tags)
      Register the flow variables needed for this client

faucet.meter module

Configure meters.

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

   Implement FAUCET configuration for an OpenFlow meter.

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

   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.

    actor_init()
        Set the LACP actor state to INIT

    actor_noact()
        Set the LACP actor state to NOACT

    actor_none()
        Set the LACP actor state to NONE

    actor_state()
        Return the current LACP actor state

    actor_state_name(state)
        Return the string of the actor state

    actor_up()
        Set the LACP actor state to UP

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

    coprocessor_defaults_types = {'strategy': <class 'str'>, 'vlan_vid_base': <class 'int'>}
    defaults = {'acl_in': None, 'acls_in': None, 'coprocessor': {}, 'count_untag_vlan_miss': {}, 'description': None, ...
    defaults_types = {'acl_in': (<class 'str'>, <class 'int'>), 'acls_in': <class 'list'>, ...

    deselect_port()
        UNSELECT the current LACP port

    finalize()
        Configuration parsing marked complete.

    get_lacp_flags()
        Get the LACP flags for the state the port is in Return sync, collecting, distributing flag values

    hosts(vlans=None)
        Return all host cache entries this port has learned (on all or specified VLANs).

    hosts_count(vlans=None)
        Return count of all hosts this port has learned (on all or specified VLANs).

    is_actor_init()
        Return true if the LACP actor state is INIT

    is_actor_noact()
        Return true if the LACP actor state is NOACT

    is_actor_none()
        Return true if the LACP actor state is NONE

    is_actor_up()
        Return true if the LACP actor state is UP

    is_port_selected()
        Return true if the lACP is a SELECTED port
**Faucet Documentation**

```python
is_port_standby()
    Return true if the lacp is a STANDBY port

is_port_unselected()
    Return true if the lacp is an UNSELECTED port

is_stack_admin_down()
    Return True if port is in ADMIN_DOWN state.

is_stack_bad()
    Return True if port is in BAD state.

is_stack_gone()
    Return True if port is in GONE state.

is_stack_init()
    Return True if port is in INIT state.

is_stack_none()
    Return True if port is in NONE state.

is_stack_up()
    Return True if port is in UP state.

lacp_port_state()
    Return the current LACP port state

lacp_update(lacp_up, now=None, lacp_pkt=None)
    Update the LACP state
    :param lacp_up: The intended LACP/port state
    :type lacp_up: bool
    :param now: Current time
    :param lacp_pkt: Received LACP packet
    Returns dyn_lacp_actor_state, dyn_lacp_current_state

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':
mirror_actions()
    Return OF actions to mirror this port.

non_stack_forwarding()
    Returns True if port is not-stacking and, and able to forward packets.

port_role_name(state)
    Return the LACP port role state name

running()
    Return True if port enabled and up.

select_port()
    SELECT the current LACP port

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

stack_admin_down()
    Change the current stack state to ADMIN_DOWN.

stack_bad()
    Change the current stack state to BAD.

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

**2.5. Source Code**
stack_descr()
   "Return stacking annotation if this is a stacking port.

stack_gone()
   Change the current stack state to GONE.

stack_init()
   Change the current stack state to INIT_DOWN.

stack_up()
   Change the current stack state to UP.

standby_port()
   Set the LACP port to STANDBY.

vlans()
   Return 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']

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

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)
   Bases: faucet.conf.Conf
   Implement FAUCET configuration for a router.

   bgp_as()
      Return BGP AS.

   bgp_connect_mode()
      Return BGP connect mode.

   bgp_defaults_types = {'as': <class 'int'>, 'connect_mode': <class 'str'>, 'neighbor_addresses': <class 'list'>, ...<class 'int'>, 'routerid': <class 'str'>, 'server_addresses': <class 'list'>, 'vlan': (<class 'str'>, <class 'int'>)}

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

   bgp_neighbor_addresses()
      Return BGP neighbor addresses.

   bgp_neighbor_addresses_by_ipv(ipv)
      Return BGP neighbor addresses with specified IP version on this VLAN.
bgp_neighbor_as()
   Return BGP neighbor AS number.

bgp_port()
   Return BGP port.

bgp_routerid()
   Return BGP router ID.

bgp_server_addresses()
   Return BGP server addresses.

bgp_server_addresses_by_ipv(ipv)
   Return BGP server addresses with specified IP version on this VLAN.

bgp_vlan()
   Return BGP VLAN.

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

defaults = {'bgp': {}, 'vlans': None}
defaults_types = {'bgp': <class 'dict'>, 'vlans': <class 'list'>}

finalize()
   Configuration parsing marked complete.

ipaddress_fields = ('neighbor_addresses', 'server_addresses')

set_bgp_vlan(vlan)
   Set BGP VLAN.

set_defaults(defaults=None, conf=None)
   Set default values and run any basic sanity checks.

vip_map(ipa)
   Return VIP for IP address, if any.

faucet.tfm_pipeline module

Configure switch tables with TFM messages.

faucet.tfm_pipeline.fill_required_properties(new_table)
   Ensure TFM has all required properties.

faucet.tfm_pipeline.init_table(table_id, name, max_entries, metadata_match, metadata_write)
   Initialize a TFM.

faucet.tfm_pipeline.load_tables(dp, valve_cl, max_table_id, min_max_flows, use_oxm_ids, fill_req)
   Configure switch tables with TFM messages.
faucet.valve module

Implementation of Valve learning layer 2/3 switch.

```python
class faucet.valve.AlliedTelesis(dp, logname, metrics, notifier, dot1x):
    Bases: faucet.valve.OVSValve
    Valve implementation for AT.
    DEC_TTL = False
    acl_manager
dot1x
    dp
    flood_manager
    host_manager
    logger
    logname
    metrics
    notifier
    ofchannel_logger
    pipeline
    recent_ofmsgs
```

```python
class faucet.valve.ArubaValve(dp, logname, metrics, notifier, dot1x):
    Bases: faucet.valve.TfmValve
    Valve implementation for Aruba.
    DEC_TTL = False
    FILL_REQ = False
    acl_manager
dot1x
dp
flood_manager
host_manager
logger
logname
metrics
notifier
ofchannel_logger
pipeline
recent_ofmsgs```
class faucet.valve.CiscoC9KValve(dp, logname, metrics, notifier, dot1x)
    Bases: faucet.valve.TfmValve
    Valve implementation for C9K.
    acl_manager
dot1x
dp
flood_manager
host_manager
logger
logname
metrics
notifier
ofchannel_logger
pipeline
recent_ofmsgs

class faucet.valve.NoviFlowValve(dp, logname, metrics, notifier, dot1x)
    Bases: faucet.valve.Valve
    Valve implementation for NoviFlow with static pipeline.
    STATIC_TABLE_IDS = True
    USE_BARRIERS = True
    acl_manager
dot1x
dp
flood_manager
host_manager
logger
logname
metrics
notifier
ofchannel_logger
pipeline
recent_ofmsgs

class faucet.valve.OVSTfmValve(dp, logname, metrics, notifier, dot1x)
    Bases: faucet.valve.TfmValve
    Valve implementation for OVS.
    MAX_TABLE_ID = 253
    MIN_MAX_FLOWS = 1000000
USE_BARRIERS = False
USE_OXM_IDS = False
acl_manager
dot1x
dp
flood_manager
host_manager
logger
logname
metrics
notifier
ofchannel_logger
pipeline
recent_ofmsgs
class faucet.valve.OVSValve(dp, logname, metrics, notifier, dot1x)
    Bases: faucet.valve.Valve
    Valve implementation for OVS.
    USE_BARRIERS = False
    acl_manager
dot1x
dp
flood_manager
host_manager
logger
logname
metrics
notifier
ofchannel_logger
pipeline
recent_ofmsgs
class faucet.valve.TfmValve(dp, logname, metrics, notifier, dot1x)
    Bases: faucet.valve.Valve
    Valve implementation that uses OpenFlow send table features messages.
    FILL_REQ = True
    MAX_TABLE_ID = 0
    MIN_MAX_FLOWS = 0
    USE_OXM_IDS = True
class faucet.valve.Valve(dp, logname, metrics, notifier, dot1x)
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
GROUPS = True
STATIC_TABLE_IDS = False
USE_BARRIERS = True

acl_manager

dot1x

dp

flood_manager

host_manager

logger

logname

metrics

notifier

ofchannel_logger

pipeline

recent_ofmsgs

class faucet.valve.Valve(dp, logname, metrics, notifier, dot1x)
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
GROUPS = True
STATIC_TABLE_IDS = False
USE_BARRIERS = True

acl_manager

add_dot1x_native_vlan(port_num, vlan_name)

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

add_vlan(vlan)
Configure a VLAN.

add_vlans(vlans)

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

close_logs()
Explicitly close any active loggers.

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

Parameters
• now(float) – current epoch time.
• discovered_up_ports(set) – datapath port numbers that are up.

Returns OpenFlow messages to send to datapath.
Return type  list

datapath_disconnect()  
Handle Ryu datapath disconnection event.

del_dot1x_native_vlan(port_num)  

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

del_vlan(vlan)  
Delete a configured VLAN.

del_vlans(vlans)

dot1x

dot1x_event(event_dict)

dp

dp_init(new_dp=None)  
Initialize datapath state at connection/re/config time.

fast_advertise(now, _other_valves)  
Called periodically to send LLDP/LACP packets.

fast_state_expire(now, other_valves)  
Called periodically to verify the state of stack ports.

flood_manager

floods_to_root()  
Return True if our dp floods (only) to root switch

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.

get_lacp_dpid_nomination(lacp_id, other_valves)  
Chooses the DP for a given LAG The DP will be nominated by the following conditions in order:

1) Number of LAG ports
2) Root DP
3) Lowest DPID

Parameters
  • lacp_id – The LACP LAG ID
  • other_valves(list) – list of other valves
Returns nominated_dpid, reason

**get_tunnel_flowmods()**
Returns flowmods for the tunnels

**host_manager**

**lacp_disable_forwarding(port, cold_start=False)**
Return OpenFlow messages for initial LACP states and when packets from the link should not be sent through the pipeline

**lacp_enable_forwarding(port)**
Enables packets from the LAG link to progress through the pipeline also recomputes the flooding & host rules

Should only be called when a LACP port is both ACTOR_UP & SELECTED Return OpenFlow messages when LACP is UP and SELECTED on a port

**lacp_handler(now, pkt_meta, other_valves)**

**lacp_update(port, lacp_up=True, now=None, lacp_pkt=None, other_valves=None, cold_start=False)**
Update the port’s LACP states and enables/disables packets from the link to be processed further through the pipeline based on the state changes

**Parameters**
- **port** – The port the packet is being received on
- **lacp_up** (bool) – Whether the lacp actor is up
- **now** (float) – The current time
- **lacp_pkt** (PacketMeta) – The received LACP packet
- **other_valves** (list) – List of other valves (in the stack)
- **cold_state** (bool) – Whether Faucet is being cold started or not

**Returns** ofmsgs

**lacp_update_actor_state(port, lacp_up, now=None, lacp_pkt=None)**

**Returns** True if LACP state changed

**Return type** bool

**lacp_update_port_selection_state(port, other_valves=None)**
Update the LACP port selection state :param port: LACP port :type port: Port :param other_valves: List of other valves :type other_valves: list

**Returns** True if port state changed

**Return type** bool

**learn_host(now, pkt_meta, other_valves)**
Possibly learn a host on a port.
Parameters

- `now (float)` – current epoch time.
- `pkt_meta (PacketMeta)` – PacketMeta instance for packet received.
- `other_valves (list)` – all Valves other than this one.

Returns OpenFlow messages, if any.

Return type list

`lldp_handler (now, pkt_meta, other_valves)`
Handle an LLDP packet.

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

`logger`

`levelname`

`metrics`

`notifier`

`notify (event_dict)`
Send an event notification.

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

`ofchannel_logger`

`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, vlan_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.
- `ekt_pkt (ryu.lib.packet.ethernet)` – parsed Ethernet header.
- `vlan_pkt (ryu.lib.packet.vlan)` – parsed VLAN 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 \((port_num, keep_cache=False, other_valves=None)\)
Return flow messages that delete port from pipeline.

port_status_handler \((port_no, reason, state, _other_valves, now)\)
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 \((port_nums, log_msg='down', keep_cache=False, other_valves=None)\)
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 by Valve.

Return type dict

recent_ofmsgs

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, _other_valves)`

Call route managers to re/resolve gateways.

**Returns** OpenFlow messages, if any by Valve.

**Return type** `dict`

`router_learn_host(pkt_meta)`

Add L3 forwarding rule.

**Parameters**

- **pkt_meta** (*PacketMeta*) – PacketMeta instance for packet received.

**Returns** OpenFlow messages, if any.

**Return type** `list`

`router_rcv_packet(now, pkt_meta)`

Process packets destined for router or run resolver.

**Parameters**

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

**Returns** OpenFlow messages.

**Return type** `list`

`router_vlan_for_ip_gw(vlan, ip_gw)`

`send_flows(ryu_dp, flow_msgs)`

Send flows to datapath (or disconnect an OF session).

**Parameters**

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

`state_expire(now, other_valves)`

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

**Parameters**

- **now** (*float*) – current epoch time.
- **other_valves** (*list*) – all Valves other than this one.

**Returns** OpenFlow messages, if any by Valve.
Return type  dict

**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 table names for configuration.

**update_metrics**(now, updated_port=None, rate_limited=False)
Update Gauge/metrics.

**update_tunnel_flowrules**()
Update tunnel ACL rules because the stack topology has changed

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

class faucet.valve_acl.ValveAclManager (port_acl_table, vlan_acl_table, egress_acl_table, pipeline, meters, dp_acls=None)
Bases: faucet.valve_manager_base.ValveManagerBase
Handle installation of ACLs on a DP

**add_authed_mac** (port_num, mac)
Add authed mac address

**add_port** (port)
Install port acls if configured

**add_port_acl** (acl, port_num, mac=None)
Create ACL openflow rules for Port

**add_vlan** (vlan)
Install vlan acls if configured

**cold_start_port** (port)
Reload acl for a port by deleting existing rules and calling add_port
create_acl_tunnel(dp)
    Create tunnel acls from ACLs that require applying in DP Returns flowmods for the tunnel :param dp: DP that contains the tunnel acls to build :type dp: DP

create_dot1x_flow_pair(port_num, nfv_sw_port_num, mac)
    Create dot1x flow pair

create_mab_flow(port_num, nfv_sw_port_num, mac)
    Create MAB ACL for sending IP Activity to Chewie NFV Returns flowmods to send all IP traffic to Chewie

Parameters
- port_num (int) – Number of port in
- nfv_sw_port_num (int) – Number of port out
- mac (str) – MAC address of the valve/port combo

del_authed_mac(port_num, mac=None, strict=True)
    remove authed mac address

del_dot1x_flow_pair(port_num, nfv_sw_port_num, mac)
    Deletes dot1x flow pair

del_mab_flow(port_num, nfv_sw_port_num, mac)
    Remove MAB ACL for sending IP Activity to Chewie NFV Returns flowmods to send all IP traffic to Chewie

Parameters
- port_num (int) – Number of port in
- nfv_sw_port_num (int) – Number of port out
- mac (str) – MAC address of the valve/port combo

del_port_acl(acl, port_num, mac=None)
    Delete ACL rules for Port

initialise_tables()
    Install dp acls if configured

faucet.valve_acl.add_mac_address_to_match(match, eth_src)
    Add or change the value of a match type

faucet.valve_acl.build_acl_entry(acl_table, rule_conf, meters, acl_allow_inst, acl_force_port_vlan_inst, port_num=None, vlan_vid=None)
    Build flow/groupmods for one ACL rule entry.

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)
    Build flow/groupmods for all entries in an ACL.

faucet.valve_acl.build_acl_port_of_msgs(acl, vid, port_num, acl_table, goto_table, priority)
    A Helper function for building Openflow Mod Messages for Port ACLs

faucet.valve_acl.build_output_actions(acl_table, output_dict)
    Implement actions to alter packet/output.
faucet.valve_acl.push_vlan\(\text{\texttt{acl_table, vlan_vid}}\)

Push a VLAN tag with optional selection of eth type.

faucet.valve_acl.rewrite_vlan\(\text{\texttt{acl_table, output_dict}}\)

Implement actions to rewrite VLAN headers.

**faucet.valve_flood module**

Manage flooding to ports on VLANs.

```python
class faucet.valve_flood.ValveFloodManager(  
    logger,  
    flood_table,  
    pipeline,  
    use_group_table,  
    groups,  
    combinatorial_port_flood,  
    combinational_port_order,  
    restricted_bcast_arpnd,  
):

    Bases: faucet.valve_manager_base.ValveManagerBase

    Implement dataplane based flooding for standalone dataplanes.
```

- FLOOD_DSTS = ((True, None, None, None), (False, None, '01:80:c2:00:00:00', 'ff:ff:ff:00:00:00'))
- RESTRICTED_FLOOD_DSTS = ((False, 2054, 'ff:ff:ff:ff:ff:ff', 'ff:ff:ff:ff:ff:ff'), (False, 34525, 'ff:ff:ff:ff:ff:ff', 'ff:ff:ff:ff:ff:ff'))

- add_vlan\(\text{\texttt{vlan}}\)
  
  install flows in response to a new VLAN

- build_flood_rules\(\text{\texttt{vlan, modify=False}}\)
  
  Add flows to flood packets to unknown destinations on a VLAN.

- del_vlan\(\text{\texttt{vlan}}\)
  
  delete flows in response to a VLAN removal

- static edge_learn_port\(\text{\texttt{\_other_valves, pkt_meta}}}\)
  
  Possibly learn a host on a port.

  **Parameters**

  - **other_valves**\(\text{\texttt{list}}\) – All Valves other than this one.
  
  - **pkt_meta**\(\text{\texttt{PacketMeta}}}\) – PacketMeta instance for packet received.

  **Returns** port to learn host on.

- floods_to_root\(\text{\texttt{\_dp_obj}}}\)
  
  Return True if the given dp floods (only) to root switch

- initialise_tables()
  
  Initialise the flood table with filtering flows.

- static update_stack_topo\(\text{\texttt{event, dp, port}}}\)
  
  Update the stack topology. It has nothing to do for non-stacking DPs.

- update_vlan\(\text{\texttt{vlan}}\)
  
  flows in response to updating an existing VLAN.
class faucet.valve_flood.ValveFloodStackManagerBase(logger, flood_table, pipeline, use_group_table, groups, combinatorial_port_flood, canonical_port_order, restricted_bcast_arpnd, stack_ports, hasExternals, dp_shortest_path_to_root, shortest_path_port, is_stack_root, is_stack_root_candidate, is_stack_edge, graph)

Bases: faucet.valve_flood.ValveFloodManager

Base class for dataplane based flooding on stacked dataplanes.

edge_learn_port (other_valves, pkt_meta)

Find a port towards the edge DP where the packet originated from

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.

shortest_path_root (edge_dp_name)

Return the port along the shortest path to/from root for edge learning

update_stack_topo (event, dp, port)

Update the stack topo according to the event.

class faucet.valve_flood.ValveFloodStackManagerNoReflection(logger, flood_table, pipeline, use_group_table, groups, combinatorial_port_flood, canonical_port_order, restricted_bcast_arpnd, stack_ports, hasExternals, dp_shortest_path_to_root, shortest_path_port, is_stack_root, is_stack_root_candidate, is_stack_edge, graph)

Bases: faucet.valve_flood.ValveFloodStackManagerBase

Stacks of size 2 - all switches directly connected to root.

Root switch simply floods to all other switches.

Non-root switches simply flood to the root.
class faucet.valve_flood.ValveFloodStackManagerReflection (logger, flood_table, pipeline, use_group_table, groups, combinatorial_port_flood, canonical_port_order, restricted_bcast_arpnd, stack_ports, hasExternals, dp_shortest_path_to_root, shortest_path_port, is_stack_root, is_stack_root_candidate, is_stack_edge, graph)

Bases: faucet.valve_flood.ValveFloodStackManagerBase

Stacks size > 2 reflect floods off of root (selective flooding).

Non-root switches flood only to the root. The root switch reflects incoming floods back out. Non-root switches flood packets from the root locally and to switches further away from the root. Flooding is entirely implemented in the dataplane.

A host connected to a non-root switch can receive a copy of its own flooded packet (because the non-root switch does not know it has seen the packet already).

A host connected to the root switch does not have this problem (because flooding is always away from the root). Therefore, connections to other non-FAUCET stacking networks should only be made to the root.

On the root switch (left), flood destinations are:
1: 2 3 4 5(s) 2: 1 3 4 5(s) 3: 1 2 4 5(s) 4: 1 2 3 5(s) 5: 1 2 3 4 5(s, note reflection)

On the middle switch:
1: 5(s) 2: 5(s) 3: 5(s) 4: 5(s) 5: 1 2 3 4 5(s) 6: 5(s)

On the rightmost switch:
1: 5(s) 2: 5(s) 3: 5(s) 4: 5(s) 5: 1 2 3 4
faucet.valve_host module

Manage host learning on VLANs.

```python
class faucet.valve_host.ValveHostFlowRemovedManager(
    logger, ports, vlans,
    eth_src_table, eth_dst_table,
    eth_dst_hairpin_table, pipeline,
    learn_timeout, learn_jitter, learn_ban_timeout,
    cache_update_guard_time, idle_dst, stack, has_externals,
    stack_root_flood_reflection)
```

Bases: faucet.valve_host.ValveHostManager

Trigger relearning on flow removed notifications.

**Note:** not currently reliable.

```python
def expire_hosts_from_vlan(_vlan, _now)
    # Expire hosts from VLAN cache.

def flow_timeout(now, table_id, match)
    # Handle a flow timed out message from dataplane.

def learn_host_timeouts(port, eth_src)
    # Calculate flow timeouts for learning on a port.
```

```python
class faucet.valve_host.ValveHostManager(
    logger, ports, vlans,
    eth_src_table, eth_dst_table,
    eth_dst_hairpin_table, pipeline,
    learn_timeout, learn_jitter, learn_ban_timeout,
    cache_update_guard_time, idle_dst, stack,
    has_externals, stack_root_flood_reflection)
```

Bases: faucet.valve_manager_base.ValveManagerBase

Manage host learning on VLANs.

```python
def add_port(port)
    # install flows in response to a new port

def add_vlan(vlan)
    # install flows in response to a new VLAN

def 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

def del_port(port)
    # delete flows in response to a port removal

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

def 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_intervlan_routing_flows (port, vlan, eth_src, eth_dst)
Returns flows for the eth_src_table that enable packets that have been routed to be accepted from an adjacent DP and then switched to the destination. Eth_src_table flow rule to match on port, eth_src, eth_dst and vlan

Parameters
• port (Port) – Port to match on.
• vlan (VLAN) – VLAN to match on
• eth_src – source MAC address (should be the router MAC)
• eth_dst – destination MAC address

learn_host_on_vlan_port_flows (port, vlan, eth_src, delete_existing, refresh_rules, 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, eth_src)
Calculate flow timeouts for learning on a port.

faucet.valve_manager_base module
Valve Manager base class

class faucet.valve_manager_base.ValveManagerBase
Bases: object

Base class for ValveManager objects.
Expected to control the installation of flows into datapath tables.
Ideally each datapath table should be controlled by 1 manager only.

add_port (port)
install flows in response to a new port

add_vlan (vlan)
install flows in response to a new VLAN

del_port (port)
delete flows in response to a port removal

del_vlan (vlan)
delete flows in response to a VLAN removal

initialise_tables ()
initialise tables controlled by this manager.

update_vlan (vlan)
flows in response to updating an existing VLAN.
faucet.valve_of module

Utility functions to parse/create OpenFlow messages.

```python
def 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

def apply_meter(meter_id)
    # Return instruction to apply a meter.
    Returns:
    Meter instruction.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPMeter

def barrier()
    # Return OpenFlow barrier request.
    Returns:
    Barrier request.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPBarrierRequest

def bucket(weight=0, watch_port=4294967295, watch_group=4294967295, actions=None)
    # Return a group action bucket with provided actions.
    Returns:
    Group action bucket.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPBucket

def build_group_flood_buckets(vlan_floodActs)
    # Return a list of group buckets to implement flooding on a VLAN.
    Returns:
    List of group buckets.
    Return type:
    list of ryu.ofproto.ofproto_v1_3_parser.OFPBucket

def build_match_dict()
    # Return dictionary of match fields.
    Returns:
    Match dictionary.
    Return type:
    dict

def controller_pps_meteradd(datapath=None, pps=0)
    # Add a PPS meter towards controller.
    Returns:
    Meter add.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPMeter

def controller_pps_meterdel(datapath=None)
    # Delete a PPS meter towards controller.
    Returns:
    Meter delete.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPMeter

def dec_ip_ttl()
    # Return OpenFlow action to decrement IP TTL.
    Returns:
    Decrement IP TTL.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPPDecNwTtl

def dedupe_ofmsgs(input_ofmsgs, random_order)
    # Return deduplicated ofmsg list.
    Returns:
    Deduplicated list.
    Return type:
    list of ryu.ofproto.ofproto_v1_3_parser.OFPOfmsg

def dedupe_output_portActs(output_portActs)
    # Deduplicate parser.OFPActionOutputs (because Ryu doesn’t define __eq__).
    Parameters:
    output_portActs (list) -- output to port actions.
    Returns:
    Output to port actions.
    Return type:
    list of ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput

def desc_stats_request(datapath=None)
    # Query switch description.
    Returns:
    Switch description.
    Return type:
    ryu.ofproto.ofproto_v1_3_parser.OFPPortDescStatsReply

def devid_present(vid)
    # Return VLAN VID without VID_PRESENT flag set.
    Parameters:
    vid (int) -- VLAN VID.
    Returns:
    VLAN VID.
    Return type:
    int
```

180 Chapter 2. Developer Documentation
Returns VLAN VID.

Return type int

faucet.valve_of.\texttt{faucet\_async}(\texttt{datapath=None, notify\_flow\_removed=False, packet\_in=True, port\_status=True})

Return async message config for FAUCET/Gauge.

faucet.valve_of.\texttt{faucet\_config}(\texttt{datapath=None})

Return switch config for FAUCET.

faucet.valve_of.\texttt{flood\_port\_outputs}(\texttt{tagged\_ports, untagged\_ports, in\_port=None, exclude\_ports=None})

Return actions for both tagged and untagged ports.

faucet.valve_of.\texttt{flood\_tagged\_port\_outputs}(\texttt{ports, in\_port=None, exclude\_ports=None})

Return list of actions necessary to flood to list of tagged ports.

faucet.valve_of.\texttt{flood\_untagged\_port\_outputs}(\texttt{ports, in\_port=None, exclude\_ports=None})

Return list of actions necessary to flood to list of untagged ports.

faucet.valve_of.\texttt{flowmod}(\texttt{cookie, command, table\_id, priority, out\_port, out\_group, match\_fields, inst, hard\_timeout, idle\_timeout, flags=0})

faucet.valve_of.\texttt{goto\_table}(\texttt{table})

Return instruction to goto table.

Parameters table (ValveTable) – table to goto.

Returns goto instruction.

Return type \texttt{ryu.ofproto.ofproto_v1\_3.parser.OFPInstruction}

faucet.valve_of.\texttt{group\_act}(\texttt{group\_id})

Return an action to run a group.

faucet.valve_of.\texttt{groupadd}(\texttt{datapath=None, type_=0, group\_id=0, buckets=None})

Add a group.

faucet.valve_of.\texttt{groupadd\_ff}(\texttt{datapath=None, group\_id=0, buckets=None})

Add a fast failover group.

faucet.valve_of.\texttt{groupdel}(\texttt{datapath=None, group\_id=4294967292})

Delete a group (default all groups).

faucet.valve_of.\texttt{groupmod}(\texttt{datapath=None, type_=0, group\_id=0, buckets=None})

Modify a group.

faucet.valve_of.\texttt{groupmod\_ff}(\texttt{datapath=None, group\_id=0, buckets=None})

Modify a fast failover group.

faucet.valve_of.\texttt{ignore\_port}(\texttt{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.\texttt{is\_apply\_actions}(\texttt{instruction})

Return True if an apply action.

Parameters instruction – OpenFlow instruction.

Returns True if an apply action.
Return type  bool

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_global_flowdel(ofmsg)
Is a delete of all flows in all tables.

faucet.valve_of.is_global_groupdel(ofmsg)
Is a delete of all groups.

faucet.valve_of.is_global_meterdel(ofmsg)
Is a delete of all meters.

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_meter(instruction)
Return True if a meter.

Parameters instruction – OpenFlow instruction.

Returns  True if a meter.

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_output(ofmsg)`

Return True if flow message is an action output message.

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

Returns True if is a OFPActionOutput.

Return type `bool`

`faucet.valve_of.is_packetout(ofmsg)`

Return True if OF message is a PacketOut

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

Returns True if is a PacketOut

Return type `bool`

`faucet.valve_of.is_set_field(action)`

`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.metadata_goto_table(metadata, mask, table)`

Return instructions to write metadata and goto table.

Parameters
• **metadata** (*int*) – metadata to write to packet
• **maks** (*int*) – mask to apply to metadata
• **table** (*ValveTable*) – table to goto.

**Returns** list of OFPInstructions

faucet.valve_of.meteradd (*meter_conf, command=0*)
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
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_non_output_actions
Split output actions into deduped actions, output ports, and non-output port actions.

  **Parameters** of ryu.ofproto.ofproto_v1_3_parser.OFPActions (list) – flood actions.

  **Returns** set of deduped actions, output ports, and non-output actions.

faucet.valve_of.output_port
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.packetouts (*port_nums, data*)
Return OpenFlow action to multiply packet out to dataplane from controller.

  **Parameters**
  • **port_num** (list) – ints, ports 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.ports_from_output_portActs*(output_portActs)*
  Return unique port numbers from OFPActionOutput actions.

  Parameters of ryu.ofproto.ofproto_v1_3_parser.OFPActionOutput *(list)* – output to port actions.

  Returns set of port number ints.

  faucet.valve_of.push_vlanAct
  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_field(**kwds**)
  Return action to set any field.

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

  Returns set field action.

  Return type ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

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

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

  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

  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, vlan_pkt, port, valve_vlan, eth_src, eth_dst, eth_type)

Bases: object

Original, and parsed Ethernet packet metadata.

ETH_TYPES_PARSERS = {2048: (4, <functools._lru_cache_wrapper object>, <class 'ryu.lib.packet.ipv4.ipv4'>), 2054: (None, None, <class 'ryu.lib.packet.arp.arp'>), 34525: (6, None, <class 'ryu.lib.packet.ipv6.ipv6'>)}

MAX_ETH_TYPE_PKT_SIZE = {2048: 174, 2054: 64}

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

data

eth_dst

eth_pkt

eth_src

eth_type

ip_ver()
    Return IP version number.

l3_dst

l3_pkt

l3_src

log()

orig_len

packet_complete()
    True if we have the complete packet.

pkt

port

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.

vlan

vlan_pkt

faucet.valve_packet.arp_reply
    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

`faucet.valve_packet.arp_request`

Return an ARP request 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)* – requested IPv4 address.

Returns serialized ARP request packet.

Return type `ryu.lib.packet.arp` faucet.valve_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.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_stack_state_tlvs(dp, port)
    Return a LLDP TLV for state of a stack port.

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.faucet_tlvs(lldp_pkt, faucet_dp_mac)
    Return list of TLVs with FAUCET OUI.

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.int_from_mac
faucet.valve_packet.int_in_mac
faucet.valve_packet.ipv4_parseable
    Return True if an IPv4 packet we could parse.

faucet.valve_packet.ipv6_link_eth_mcast
    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
    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_actor_up(lacp_pkt)
    Return 1 if remote LACP link is up.

faucet.valve_packet.lacp_reqreply
    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.
- **actor_state_synchronization** *(int)* – 1 if we will use this link.
- **actor_state_activity** *(int)* – 1 if actively sending LACP.
- **actor_state_collecting** *(int)* – 1 if receiving on this link.
- **actor_state_distributing** *(int)* – 1 if transmitting on this link.
- **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, portdescr=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.

2.5. Source Code
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faucet.valve_packet.mac_addr_all_zeros</td>
<td>Returns True if mac_addr is all zeros.</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>mac_addr (str) – MAC address.</td>
<td></td>
</tr>
<tr>
<td>Returns</td>
<td></td>
</tr>
<tr>
<td>True if all zeros.</td>
<td></td>
</tr>
<tr>
<td>Return type</td>
<td>bool</td>
</tr>
</tbody>
</table>

| faucet.valve_packet.mac_addr_is_unicast | 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.mac_mask_bits | Return number of bits in MAC mask or 0. |

| faucet.valve_packet.nd_advert | 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 | Return IPv6 neighbor discovery request packet. |
| Parameters |
| vid (int or None) – VLAN VID to use (or None). |
| eth_src (str) – source Ethernet MAC address. |
| eth_dst (str) – Ethernet destination 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_faucet_lldp(lldp_pkt, faucet_dp_mac)
    Parse and return FAUCET TLVs from LLDP packet.

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
    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.router_advert(vid, eth_src, eth_dst, src_ip, dst_ip, vips, pi_flags=6)
    Return IPv6 ICMP Router Advert.

    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_packet.tlv_cast(tlvs, tlv_attr, cast_func)
    Return cast’d attribute of first TLV or None.

faucet.valve_packet.tlvs_by_subtype(tlvs, subtype)
    Return list of TLVs with matching type.

faucet.valve_packet.tlvs_by_type(tlvs, tlv_type)
    Return list of TLVs with matching type.
faucet.valve_pipeline module

Manages movement of packets through the faucet pipeline.

```python
class faucet.valve_pipeline.ValvePipeline(dp):
    Bases: faucet.valve_manager_base.ValveManagerBase
```

Responsible for maintaining the integrity of the Faucet pipeline for a single valve.

Controls what packets a module sees in its tables and how it can pass packets through the pipeline.

Responsible for installing flows in the vlan, egress and classification tables

**accept_to_classification** *(actions=None)*

Get instructions to forward packet through the pipeline to classification table.

- **Parameters**
  - `actions`: (optional) list of actions to apply to packet.

- **Returns**
  - list of instructions

**accept_to_egress** *(actions=None)*

Get instructions to forward packet through the pipeline to egress table

- **Raises**
  - AssertionError if egress pipeline is not configured

- **Parameters**
  - `actions`: (optional) list of actions to apply to the packet

- **Returns**
  - list of instructions

**accept_to_l2_forwarding** *(actions=None)*

Get instructions to forward packet through the pipeline to l2 forwarding.

- **Parameters**
  - `actions`: (optional) list of actions to apply to packet.

- **Returns**
  - list of instructions

**accept_to_vlan** *(actions=None)*

Get instructions to forward packet through the pipeline to vlan table.

- **Parameters**
  - `actions`: (optional) list of actions to apply to packet.

- **Returns**
  - list of instructions

**add_port** *(port)*

install flows in response to a new port

**del_port** *(port)*

delete flows in response to a port removal

**filter_packets** *(match_dict, priority_offset=0)*

get a list of flow modification messages to filter packets from the pipeline.

- **Parameters**
  - `match_dict`: a dictionary specifying the match fields
  - `priority_offset`: used to prevent overlapping entries

**initialise_tables** ()

Install rules to initialise the classification table

**output** *(port, vlan, hairpin=False, external_forwarding_requested=None)*

Get instructions list to output a packet through the regular pipeline.

- **Parameters**
  - `port`: Port object of port to output packet to
  - `vlan`: Vlan object of vlan to output packet on
  - `hairpin`: if True, hairpinning is required
• **apply_egress_acl** – if True the packet will be sent to the egress acl table before being output

**Returns** list of Instructions

`remove_filter (match_dict, strict=True, priority_offset=0)`  
retrieve flow mods to remove a filter from the classification table

`select_packets (target_table, match_dict, actions=None, priority_offset=0)`  
retrieve rules to redirect packets matching match_dict to table

**faucet.valve_route module**

Valve IPv4/IPv6 routing implementation.

```python
class faucet.valve_route.AnonVLAN(vid)
    Bases: object

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.

    cache_time

    dead (max_fib_retries)
    Return True if this nexthop is considered dead.

    eth_src

    last_retry_time

    next_retry (now, max_resolve_backoff_time)
    Increment state for next retry.

    next_retry_time

    port

    resolution_due (now, max_age)
    Return True if this nexthop is due to be re resolved/retried.

    resolve_retries
```

```python
class faucet.valve_route.ValveIPv4RouteManager (logger, notify, global_vlan,
    neighbor_timeout, max_hosts_perResolve_cycle, max_host_fib_retry_count,
    max_resolve_backoff_time, proactive_learn, dec_ttl, multi_out,
    fib_table, vip_table, pipeline, routers, flood_manager)

    Bases: faucet.valve_route.ValveRouteManager
```

Implement IPv4 RIB/FIB.

`CONTROL_ETH_TYPES = (2048, 2054)`

`ETH_TYPE = 2048`

`ICMP_SIZE = 174`
ICMP_TYPE = 1
IPV = 4
IP_PKT
    alias of ryu.lib.packet.ipv4.ipv4
active
advertise(_vlan)
control_plane_handler(now, pkt_meta)
    Handle packets destined for router otherwise proactively learn host information
dec_ttl
fib_table
flood_manager
global_routing
global_vlan
logger
max_host_fib_retry_count
max_hosts_per_resolve_cycle
max_resolve_backoff_time
multi_out
neighbor_timeout
notify
pipeline
proactive_learn
route_priority
routers
vip_table

class faucet.valve_route.ValveIPv6RouteManager(logger, notify, global_vlan,
    neighbor_timeout,
    max_hosts_per_resolve_cycle,
    max_host_fib_retry_count,
    max_resolve_backoff_time, proactive_learn, dec_ttl, multi_out,
    fib_table, vip_table, pipeline, routers, flood_manager)

Bases: faucet.valve_route.ValveRouteManager
Implement IPv6 FIB.

CONTROL_ETH_TYPES = (34525,)
ETH_TYPE = 34525
ICMP_SIZE = 194
ICMP_TYPE = 58
IPV = 6
IP_PKT
    alias of ryu.lib.packet.ipv6.ipv6
active
advertise(vlan)
control_plane_handler(now, pkt_meta)
    Resolve packets destined for router or proactively learn host information
dec_ttl
fib_table
flood_manager
global_routing
global_vlan
logger
max_host_fib_retry_count
max_hosts_per_resolve_cycle
max_resolve_backoff_time
multi_out
neighbor_timeout
notify
pipeline
proactive_learn
route_priority
routers
vip_table
class faucet.valve_route.ValveRouteManager(logg
    er, notify, global_vlan, neighbor_timeout, max_hosts_per_resolve_cycle,
    max_host_fib_retry_count,
    max_resolve_backoff_time, proactive_learn, dec_ttl, multi_out, fib_table, vip_table,
    pipeline, routers, flood_manager)
Bases: faucet.valve_manager_base.ValveManagerBase
Base class to implement RIB/FIB.
CONTROL_ETH_TYPES = ()
ETH_TYPE = None
ICMP_SIZE = None
ICMP_TYPE = None
IPV = 0
IP_PKT = None
MAX_PACKET_IN_SIZE = 194
active
add_host_fib_route_from_pkt \((\text{now, pkt\_meta})\)
Add a host FIB route given packet from host.

Parameters
- \text{now} (\text{float}) – seconds since epoch.
- \text{pkt\_meta} (\text{PacketMeta}) – received packet.

Returns OpenFlow messages.

Return type list

add_route \((\text{vlan, ip\_gw, ip\_dst})\)
Add a route to the RIB.

Parameters
- \text{vlan} (\text{vlan}) – VLAN containing this RIB.
- \text{ip\_gw} (\text{ipaddress.ip\_address}) – IP address of nexthop.
- \text{ip\_dst} (\text{ipaddress.ip\_network}) – destination IP network.

Returns OpenFlow messages.

Return type list

add_vlan \((\text{vlan})\)
install flows in response to a new VLAN

advertise \((\text{vlan})\)
control\_plane\_handler \((\text{now, pkt\_meta})\)

dec\_ttl

del_route \((\text{vlan, ip\_dst})\)
Delete a route from the RIB.

Only one route with this exact destination is supported.

Parameters
- \text{vlan} (\text{vlan}) – VLAN containing this RIB.
- \text{ip\_dst} (\text{ipaddress.ip\_network}) – destination IP network.

Returns OpenFlow messages.

Return type list

expire\_port\_nexthops \((\text{port})\)
Expire all hosts on a port

fib\_table
flood\_manager
global\_routing
global\_vlan
logger
max_host\_fib\_retry\_count
max_hosts\_per\_resolve\_cycle
max\_resolve\_backoff\_time
multi_out
neighbor_timeout
nexthop_dead(nexthop_cache_entry)
Returns true if the nexthop_cache_entry is considered dead
notify
notify_learn(pkt_meta)
pipeline
proactive_learn
resolve_expire_hosts(vlan, now, resolve_all=True)
Re/resolve hosts.
Parameters
• vlan(vlan) – VLAN containing this RIB/FIB.
• now(float) – seconds since epoch.
• resolve_all(bool) – attempt to resolve all unresolved gateways.
Returns OpenFlow messages.
Return type list
resolve_gateways(vlan, now, resolve_all=True)
Re/resolve gateways.
Parameters
• vlan(vlan) – VLAN containing this RIB/FIB.
• now(float) – seconds since epoch.
• resolve_all(bool) – attempt to resolve all unresolved gateways.
Returns OpenFlow messages.
Return type list
route_priority
router_vlan_for_ip_gw(vlan, ip_gw)
Return router VLAN for IP gateway (or None).
Parameters
• vlan(vlan) – VLAN containing this RIB.
• ip_gw(ipaddress.ip_address) – IP address of nexthop.
Returns VLAN for this gateway or None.
routers
vip_table
Faucet Documentation

faucet.valve_ryuapp module

RyuApp base class for FAUCET/Gauge.

```python
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.

    OFP_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.

exception faucet.valve_ryuapp.ValveDeadThreadException
    Bases: Exception
    Exception raised when a dead thread is detected.
```

faucet.valve_table module

Abstraction of an OF table.

```python
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)
Update entry with new buckets.

class faucet.valve_table.ValveGroupTable
Bases: object
Wrap access to group table.

def delete_all()
Delete all groups.

entries = None
get_entry(group_id, buckets)
Update entry with group_id with buckets, and return the entry.

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

class faucet.valve_table.ValveTable(name, table_config, flow_cookie, notify_flow_removed=False, next_tables=None)
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.

goto(next_table)
Add goto next table instruction.

goto_miss(next_table)
Add miss goto table instruction.

goto_this()

match
Compose an OpenFlow match rule.

set_external_forwarding_requested()
Set field for external forwarding requested.

set_field(**kwds)
Return set field action.

set_no_external_forwarding_requested()
Set field for no external forwarding requested.

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

Parameters

vid (int) – VLAN VID

2.5. Source Code 199
Returns  set VID with VID_PRESENT.

Return type  ryu.ofproto.ofproto_v1_3_parser.OFPActionSetField

faucet.valve_util module

Utility functions for FAUCET.

faucet.valve_util.close_logger(logger)
    Close all handlers on logger object.

faucet.valve_util.dpid_log(dp_id)
    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.valve_util.utf8_decode(msg_str)
    Gracefully decode a possibly UTF-8 string.

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.MetaDPState
    Bases: object
    Contains state/config about all DPs.
class faucet.valves_manager.ValvesManager (logname, logger, metrics, notifier, bgp, dot1x,
config_auto_revert, send_flows_to_dp_by_id)

Bases: object

Manage a collection of Valves.

datapath_connect (now, valve, discovered_up_ports)
  Handle connection from DP.

event_socket_heartbeat (now)
  raises event for event sock heartbeat

healthy_stack_roots (now, candidate_dps)
  Return list of healthy stack root names.

load_configs (now, new_config_file, delete_dp=None)
  Load/apply new config to all Valves.

maintain_stack_root (now)
  Maintain current stack root and return True if stack root changes.

new_valve (new_dp)

parse_configs (new_config_file)
  Return parsed configs for Valves, or None.

port_status_handler (valve, msg, now)
  Handle a port status change message.

request_reload_configs (now, new_config_file, delete_dp=None)
  Process a request to load config changes.

revert_config ()
  Attempt to revert config to last known good version.

update_config_applied (sent=None, reset=False)
  Update faucet_config_applied from {dpid: sent} dict, defining applied == sent == enqueued via Ryu

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, msg)
  Time a call to Valve packet in handler.

valves = None

faucet.vlan module

VLAN configuration.

class faucet.vlan.AnyVLAN
  Bases: object

  Placeholder any tagged VLAN.

  name = 'Any VLAN'

  vid = 4096
class faucet.vlan.HostCacheEntry (eth_src, port, cache_time)
    Bases: object
    Association of a host with a port.
    cache_time
    eth_src
    eth_src_int
    port
class faucet.vlan.NullVLAN
    Bases: object
    Placeholder null VLAN.
    name = 'Null VLAN'
    vid = 0
class faucet.vlan.OFVLAN (name, vid)
    Bases: object
class faucet.vlan.VLAN (_id, dp_id, conf=None)
    Bases: faucet.conf.Conf
    Contains state for one VLAN, including its configuration.
    add_cache_host (eth_src, port, cache_time)
        Add/update a host to the cache on a port at at time.
    add_route (ip_dst, ip_gw)
        Add an IP route.
    all_ip_gws (ipv)
        Return all IP gateways for specified IP version.
    cached_host (eth_src)
        Return host from cache or None.
    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, 'acl_out': None, 'acls_in': None, 'acls_out': None, 'description': None, 'dot1x_assigned': ... 0, 'reserved_internal_vlan': False, 'routes': None, 'targeted_gw_resolution': True, 'unicast_flood': True, 'vid': None}
    defaults_types = {'acl_in': (<class 'int'>, <class 'str'>), 'acl_out': (<class 'int'>, <class 'str'>), 'acls_in': <class 'list'>, 'targeted_gw_resolution': <class 'bool'>, 'unicast_flood': <class 'bool'>, 'vid': <class 'int'>}
    del_route (ip_dst)
        Delete an IP route.
    exclude_native_if_dot1x ()
        Don’t output on native vlan, if dynamic (1x) vlan is in use
excluded_lag_ports(in_port=None)
Ensure output to SELECTED LAG ports & only one LAG member

expire_cache_host(eth_src)
Expire a host from caches.

expire_cache_hosts(now, learn_timeout)
Expire stale host entries.

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

flood_pkt(packet_builder, multi_out=True, *args)
Return Packet-out actions via flooding

static flood_ports(configured_ports, exclude_unicast)
Return configured ports that allow flooding

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 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, faucet_vip=None)
Return faucet_vip if IP in same IP network as a VIP on this VLAN.

ipvs()
Return IP versions configured on this VLAN.

is_faucet_vip(ipa, faucet_vip=None)
Return True if IP is a VIP on this VLAN.

is_host_fib_route(host_ip)
Return True if IP destination is a host FIB route.

Parameters host_ip (ipaddress.ip_address): potential host FIB route.

Returns True if a host FIB route (and not used as a gateway).

lacp_ports()
Return ports that have LACP on this VLAN.

lacp_up_selected_ports()
Return LACP ports that have been SELECTED and are UP

lags()
Return dict of LAGs mapped to member ports.
link_and_other_vips (ipv)
    Return link local and non-link local VIPs.

loop_protect_external_ports()
    Return ports with external loop protection set.

loop_protect_external_ports_up()
    Return up ports with external loop protection set.

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

mutableAttrs = frozenset({'dot1x_untagged', 'tagged', 'untagged'})

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

neigh_cache_count_by_ipv (ipv)
    Return number of hosts in neighbor cache for specified IP version on this VLAN.

output_port (port, hairpin=False, output_table=None, external_forwarding_requested=None)

pkt_out_port (packet_builder, port, *args)
    Return packet-out actions with VLAN tag if port is tagged

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.

reset_caches()
    Reset dynamic caches.

reset_ports (ports)
    Reset tagged and untagged port lists.

restricted_bcast_arpnd_ports()
    Return all ports with restricted broadcast enabled.

route_count_by_ipv (ipv)
    Return route table count for specified IP version on this VLAN.

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

selected_up_lags()
    Return dict of LAGs mapped to member ports that have been selected

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

tagged_flood_ports (exclude_unicast)

untagged_flood_ports (exclude_unicast)

static_vld_valid (vid)
    Return True if VID valid.

vip_map (ipa)
    Return the vip containing ipa
faucet.watcher module

Gauge watcher implementations.

```python
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

class faucet.watcher.GaugeMeterStatsLogger (conf, logname, prom_client)
    Bases: faucet.gauge_pollers.GaugeMeterStatsPoller

    Abstraction for meter statistics logger.

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.

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

    Abstraction for port statistics logger.

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.

```python
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.
- **path** (string): path where files should be written when writing to multiple files
- **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.

**check_config()**

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

```
db_defaults = {'compress': False, 'file': None, 'influx_db': 'faucet', 'influx_host': 'localhost', 'influx_port': 8086, ... '', 'path': None, 'prometheus_addr': '0.0.0.0', 'prometheus_port': 9303, 'prometheus_test_thread': False, 'type': None}
db_defaults_types = {'compress': <class 'bool'>, 'file': <class 'str'>, 'influx_db': <class 'str'>, 'influx_host': <class 'str'>, 'influx_port': <class 'int'>, 'influx_user': <class 'str'>, 'influx_pwd': <class 'str'>, 'influx_timeout': <class 'int'>, 'influx_retries': <class 'int'>, 'prometheus_addr': <class 'str'>, 'prometheus_port': <class 'int'>, 'prometheus_test_thread': <class 'bool'>, 'type': <class 'str'>}
defaults = {'all_dps': False, 'db': None, 'db_type': 'text', 'dbs': None, 'dps': None, 'interval': 30, 'name': None, 'type': None}
defaults_types = {'all_dps': <class 'bool'>, 'db': <class 'str'>, 'db_type': <class 'str'>, 'dbs': <class 'list'>, 'dps': <class 'list'>, 'interval': <class 'int'>, 'name': <class 'str'>, 'type': <class 'str'>}
```
Module contents
CHAPTER
THREE

INDICES AND TABLES

- genindex
- modindex
- search
faucet, 207
faucet.acl, 140
faucet.check_faucet_config, 142
faucet.conf, 142
faucet.config_parser, 143
faucet.config_parser_util, 144
faucet.dp, 144
faucet.faucet, 148
faucet.faucet_bgp, 150
faucet.faucet_dot1x, 150
faucet.faucet_event, 152
faucet.faucet_experimental_api, 152
faucet.faucet_metadata, 153
faucet.faucet_metrics, 153
faucet.faucet_pipeline, 153
faucet.fctl, 153
faucet.gauge, 154
faucet.gauge_influx, 155
faucet.gauge_pollers, 157
faucet.gauge_prom, 159
faucet.meter, 159
faucet.port, 160
faucet.prom_client, 162
faucet.router, 162
faucet.tfm_pipeline, 163
faucet.valve, 164
faucet.valve_acl, 173
faucet.valve_flood, 175
faucet.valve_host, 178
faucet.valve_manager_base, 179
faucet.valve_of, 180
faucet.valve_of_old, 186
faucet.valve_packet, 186
faucet.valve_pipeline, 192
faucet.valve_route, 193
faucet.valve_ryuapp, 198
faucet.valve_table, 198
faucet.valve_util, 200
faucet.valves_manager, 200
faucet.vlan, 201
faucet.watcher, 205
faucet.watcher_conf, 205
INDEX

A

accept_to_classification() (faucet.valve_pipeline.ValvePipeline method), 192
accept_to_egress() (faucet.valve_pipeline.ValvePipeline method), 192
accept_to_l2_forwarding() (faucet.valve_pipeline.ValvePipeline method), 192
accept_to_vlan() (faucet.valve_pipeline.ValvePipeline method), 192
ACL (class in faucet.acl), 140
acl_manager (faucet.valve.AlliedTelesis attribute), 164
cacl_manager (faucet.valve.ArubaValve attribute), 164
cacl_manager (faucet.valve.CiscoC9KValve attribute), 165
cacl_manager (faucet.valve.NoviFlowValve attribute), 165
cacl_manager (faucet.valve.OVSTfmValve attribute), 166
cacl_manager (faucet.valve.OVSValve attribute), 166
cacl_manager (faucet.valve.Valve attribute), 167
acquire_nonblock() (faucet.faucet_event.NonBlockLock method), 152
actions_types (faucet.acl.ACL attribute), 141
active (faucet.valve_route.ValveIPv4RouteManager attribute), 194
active (faucet.valve_route.ValveIPv6RouteManager attribute), 195
active (faucet.valve_route.ValveRouteManager attribute), 195
actor_init() (faucet.port.Port method), 160
actor_noact() (faucet.port.Port method), 160
actor_none() (faucet.port.Port method), 160
actor_state() (faucet.port.Port method), 160
actor_state_name() (faucet.port.Port method), 160
actor_up() (faucet.port.Port method), 160
add() (faucet.valve_table.ValveGroupEntry method), 198
add_acl() (faucet.dp.DP method), 144
add_authed_mac() (faucet.valve_acl.ValveAclManager method), 173
add_cache_host() (faucet.vlan.VLAN method), 202
add_db() (faucet.watcher_conf.WatcherConf method), 206
add_dot1x_native_vlan() (faucet.valve.Valve method), 167
add_dp() (faucet.watcher_conf.WatcherConf method), 206
add_host_fib_route_from_pkt() (faucet.valve_route.ValveRouteManager method), 195
add_mac_address_to_match() (in module faucet.valve_acl), 174
add_port() (faucet.dp.DP method), 144
add_port() (faucet.valve_acl.ValveAclManager method), 173
add_port() (faucet.valve_host.ValveHostManager method), 178
add_port() (faucet.valve_manager_base.ValveManagerBase method), 179
add_port() (faucet.valve_pipeline.ValvePipeline method), 192
add_port_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
add_port_acl() (faucet.valve_acl.ValveAclManager method), 173
add_route() (faucet.valve.Valve method), 167
add_route() (faucet.valve_route.ValveRouteManager method), 196
add_route() (faucet.vlan.VLAN method), 202
add_router() (faucet.dp.DP method), 144
add_stack_link() (faucet.dp.DP class method), 144
add_vlan() (faucet.valve.Valve method), 167
add_vlan() (faucet.valve_acl.ValveAclManager method), 173
add_vlan() (faucet.valve_flood.ValveFloodManager method), 175

213
Index

add_vlan() (faucet.valve_host.ValveHostManager method), 178
add_vlan() (faucet.valve_manager_base.ValveManagerBase method), 179
add_vlan() (faucet.valve_route.ValveRouteManager method), 196
add_vlan_acl() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
add_vlans() (faucet.valve.Valve method), 167
advertise() (faucet.valve.Valve method), 167
advertise() (faucet.valve_route.ValveIPv4RouteManager method), 194
advertise() (faucet.valve_route.ValveIPv6RouteManager method), 195
advertise() (faucet.valve_route.ValveRouteManager method), 196
age() (faucet.valve_route.NextHop method), 193
all_ip_gws() (faucet_vlan.VLAN method), 202
all_lags_up() (faucet.dp.DP method), 145
AlliedTelesis (class in faucet.valve), 164
AnonVLAN (class in faucet.valve.route), 193
any_stack_port_up() (faucet.dp.DP method), 145
AnyVLAN (class in faucet.vlan), 201
apply_actions() (in module faucet.valve_of), 180
apply_meter() (in module faucet.valve_of), 180
arp_reply (in module faucet.valve_packet), 186
arp_request (in module faucet.valve.packet), 187
ArubaValve (class in faucet.valve), 164
auth_handler() (faucet.faucet_dot1x.FaucetDot1x method), 150

B
ban_rules() (faucet.valve_host.ValveHostManager method), 178
barrier (in module faucet.valve_of), 180
base_prom_labels() (faucet.dp.DP method), 145
bgp (faucet.faucet.Faucet attribute), 148
bgp_as() (faucet.router.Router method), 162
bgp_connect_mode() (faucet.router.Router method), 162
bgp_defaults_types (faucet.router.Router attribute), 162
bgp_ipvs() (faucet.router.Router method), 162
bgp_neighbor_addresses() (faucet.router.Router method), 162
bgp_neighbor_addresses_by_ipv() (faucet.router.Router method), 162
bgp_neighbor_as() (faucet.router.Router method), 162
bgp_port() (faucet.router.Router method), 163
bgp_routerid() (faucet.router.Router method), 163
bgp_routers() (faucet.dp.DP method), 145
bgp_server_addresses() (faucet.router.Router method), 163
bgp_server_addresses_by_ipv() (faucet.router.Router method), 163
BgpSpeakerKey (class in faucet.faucet_bgp), 150
bucket() (in module faucet.valve_of), 180
build() (faucet_acl.ACL method), 141
build_acl_ofmsgs() (in module faucet.valve_acl), 174
build_acl_port_of_msgs() (in module faucet.valve_acl), 174
build_flood_rules() (faucet_valve_flood.ValveFloodManager method), 175
build_group_flood_buckets() (in module faucet_valve_of), 180
build_match_dict (in module faucet.valve_of), 180
build_output_dict (in module faucet_valve_of), 180
build_pkt_actions() (in module faucet_valve_packet), 187

C
C

214
clear_cache_hosts_on_port() (faucet.vlan.VLAN method), 202
clone_dyn_state() (faucet.dp.DP method), 145
close_logger() (in module faucet.valve_util), 200
close_logs() (faucet.valve.Valve method), 167
cold_start() (faucet.dp.DP method), 145
cold_start_port() (faucet.valve_acl.ValveAclManager method), 173
Conf (class in faucet.conf), 142
conf (faucet.gauge_influx.InfluxShipper attribute), 157
conf_diff() (faucet.conf.Conf method), 142
conf_hash() (faucet.conf.Conf method), 142
config_changed() (in module faucet.config_parser_util), 144
config_file (faucet.valves_manager.ConfigWatcher attribute), 200
config_file_hash() (in module faucet.config_parser_util), 144
config_file_stats (faucet.valves_manager.ConfigWatcher attribute), 200
config_hashes (faucet.valves_manager.ConfigWatcher attribute), 200
ConfigWatcher (class in faucet.valves_manager), 200
connect_or_disconnect_handler() (faucet.valve_ryuapp.RyuAppBase method), 198
construct_mapping() (faucet.config_parser_util.UniqueKeyLoader method), 144
content_changed() (faucet.valves_manager.ConfigWatcher method), 200
CONTROL_ETH_TYPES (faucet.valve_route.ValveIPv4RouteManager attribute), 193
CONTROL_ETH_TYPES (faucet.valve_route.ValveIPv6RouteManager attribute), 194
CONTROL_ETH_TYPES (faucet.valve_route.ValveRouteManager attribute), 195
control_plane_handler() (faucet.valve_route.ValveIPv4RouteManager method), 194
control_plane_handler() (faucet.valve_route.ValveIPv6RouteManager method), 195
control_plane_handler() (faucet.valve_route.ValveRouteManager method), 196
controller_pps_meteradd() (in module faucet.valve_of), 180
core_controller_pps_meterdel() (in module faucet.valve_of), 180
coprocessor_defaults_types (faucet.port.Port attribute), 160
coprocessor_ports() (faucet.dp.DP method), 145
create_acl_tunnel() (faucet.valve_acl.ValveAclManager method), 174
create_dot1x_flow_pair() (faucet.valve_acl.ValveAclManager method), 174
create_flow_pair() (faucet.valve_ryuapp.RyuAppBase method), 198
create_mab_flow() (faucet.valve_ryuapp.RyuAppBase method), 198
create_mab_flow() (faucet.valve_acl.ValveAclManager method), 174
D
data (faucet.valve_packet.PacketMeta attribute), 186
datatapath_connect() (faucet.valve.Valve method), 167
datatapath_connect() (faucet.valves_manager.ValvesManager method), 201
datatapath_disconnect() (faucet.valve.Valve method), 168
db_defaults (faucet.watcher_conf.WatcherConf attribute), 206
db_defaults_types (faucet.watcher_conf.WatcherConf attribute), 206
dead() (faucet.valve_route.NextHop method), 193
debug() (faucet.valve.ValveLogger method), 173
dec_ip_ttl() (in module faucet.valve_of), 180
DEC_TTL (faucet.valve.AlliedTelesis attribute), 180
DEC_TTL (faucet.valve.ArubaValve attribute), 164
DEC_TTL (faucet.valve.Valve attribute), 167
dec_ttl (faucet.valve_route.ValveIPv4RouteManager attribute), 194
dec_ttl (faucet.valve_route.ValveIPv6RouteManager attribute), 195
dec_ttl (faucet.valve_route.ValveRouteManager attribute), 196
dec_ttl (faucet.valve_route.ValveRouteManager attribute), 195
dec_ttl (faucet.valve_route.ValveRouteManager attribute), 196
decode_value() (in module faucet.fctl), 153
dedupe_ofmsgs() (in module faucet.valve_of), 180
dedupe_output_portActs() (in module faucet.valve_of), 180
DEFAULT_LLDP_MAX_PER_INTERVAL (faucet.dp.DP attribute), 144

Index 215
Faucet Documentation

216 Index

defaults

defaults

default_table_sizes_types (faucet.dp.DP attribute), 145
defauls_types (faucet.acl.ACL attribute), 141
defaults (faucet.conf.Conf attribute), 142
defaults (faucet.dp.DP attribute), 145
defaults (faucet.meter.Meter attribute), 159
defaults (faucet.port.Port attribute), 160
defaults (faucet.router.Router attribute), 163
defaults (faucet.vlan.VLAN attribute), 202
defaults (faucet.watcher_conf.WatcherConf attribute), 206
defaults_types (faucet.acl.ACL attribute), 141
defaults_types (faucet.conf.Conf attribute), 143
defaults_types (faucet.dp.DP attribute), 145
defaults_types (faucet.meter.Meter attribute), 159
defaults_types (faucet.port.Port attribute), 160
defaults_types (faucet.router.Router attribute), 163
defaults_types (faucet.vlan.VLAN attribute), 202
defaults_types (faucet.watcher_conf.WatcherConf attribute), 206
del_authed_mac() (faucet.valve_acl.ValveAclManager method), 174
del_dot1x_flow_pair() (faucet.valve_acl.ValveAclManager method), 174
del_dot1x_nativ Krankenhaus (faucet.valve.Valve method), 148
del_mab_flow() (faucet.valve_acl.ValveAclManager method), 174
del_port() (faucet.valve.host.ValveHostManager method), 178
del_port() (faucet.valve_manager_base.ValveManagerBase method), 179
del_port() (faucet.valve_pipeline.ValvePipeline method), 192
del_port_acl() (faucet.valve_acl.ValveAclManager method), 174
del_route() (faucet.valve.Valve method), 168
del_route() (faucet.valve_route.ValveRouteManager method), 196
del_vlan() (faucet.vlan.VLAN method), 202
del_vlan() (faucet.valve.Valve method), 168
del_vlan() (faucet.valve_flood.ValveFloodManager method), 175
del_vlan() (faucet.valve_manager_base.ValveManagerBase method), 179
del_vlans() (faucet.valve.Valve method), 168
delete() (faucet.valve_table.ValveGroupEntry method), 198
delete_all() (faucet.valve_table.ValveGroupTable method), 199
 delete_host_from_vlan() (faucet.valve_host.ValveHostManager method), 178
delete_port_acl() (faucet.faustor_experimental_api.FaucetExperimentalAPI method), 152
delete_vlan_acl() (faucet.faustor_experimental_api.FaucetExperimentalAPI method), 152
desc_stats_reply_handler() (faucet.valve.Faucet method), 148
desc_stats_request() (in module faucet.valve, 180
deselect_port() (faucet.port.Port method), 160
defid_present() (in module faucet.valve_of), 180
dot1x() (faucet.valve.AlliedTelesis attribute), 164
dot1x() (faucet.valve.ArubaValve attribute), 164
dot1x() (faucet.valve.CiscoC9KValve attribute), 165
dot1x() (faucet.valve.NoviFlowValve attribute), 165
dot1x() (faucet.valve.OVSTfmValve attribute), 166
dot1x() (faucet.valve.OVSValve attribute), 166
dot1x() (faucet.valve.TfmValve attribute), 167
dot1x() (faucet.valve.Valve attribute), 168
dot1x_event() (faucet.valve.Valve method), 168
dot1x_ports() (faucet.dp.DP method), 145
dp() (faucet.valve.AlliedTelesis attribute), 164
dp() (faucet.valve.ArubaValve attribute), 164
dp() (faucet.valve.CiscoC9KValve attribute), 165
dp() (faucet.valve.NoviFlowValve attribute), 165
dp() (faucet.valve.OVSTfmValve attribute), 166
dp() (faucet.valve.OVSValve attribute), 166
dp() (faucet.valve.TfmValve attribute), 167
dp() (faucet.valve.Valve attribute), 168
dp_config_path() (in module faucet.valve_config_parser_util),
dp_include() (in module faucet.valve_config_parser_util),
dp_init() (faucet.valve.Valve method), 168
dp_parser() (in module faucet.valve_config_parser), 143
dp_preparsed_parser() (in module faucet.valve_config_parser), 143
dpid_log() (in module faucet.valve_util), 200
dyn_finalized() (faucet.conf.Conf attribute), 143
dyn_hash() (faucet.conf.Conf attribute), 143
dyn_hash() (faucet.valve_flood.ValveFloodManager static method), 175
dyn_hash() (faucet.valve_flood.ValveFloodStackManagerBase
flowmod() (faucet.valve_table.ValveTable method), 199
flowmod() (in module faucet.valve.of), 181
flowremoved_handler() (faucet.faucet.Faucet method), 149
from_connected_to_vip() (faucet.vlan.VLAN method), 203

G
Gauge (class in faucet.gauge), 154
GaugeFlowTableInfluxDBLogger (class in faucet.gauge.influx), 155
GaugeFlowTableLogger (class in faucet.watcher), 205
GaugeFlowTablePoller (class in faucet.gauge.pollers), 157
GaugeFlowTablePrometheusPoller (class in faucet.gauge.prom), 159
GaugeMeterStatsLogger (class in faucet.watcher), 205
GaugeMeterStatsPoller (class in faucet.gauge.pollers), 157
GaugeMeterStatsPrometheusPoller (class in faucet.gauge.prom), 159
GaugePoller (class in faucet.gauge.pollers), 157
GaugePortStateInfluxDBLogger (class in faucet.gauge.influx), 155
GaugePortStateLogger (class in faucet.watcher), 205
GaugePortStatePoller (class in faucet.gauge.pollers), 158
GaugePortStatePrometheusPoller (class in faucet.gauge.prom), 159
GaugePortsInfluxDBLogger (class in faucet.gauge.influx), 156
GaugePortsLogger (class in faucet.watcher), 205
GaugePortsPoller (class in faucet.gauge.pollers), 158
GaugePortsPrometheusPoller (class in faucet.gauge.prom), 159
GaugePrometheusClient (class in faucet.gauge.prom), 159
GaugeThreadPoller (class in faucet.gauge.pollers), 158
get_config() (faucet.faucet.Faucet method), 149
get_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
get_config_changes() (faucet.dp.DP method), 145
get_config_dict() (faucet.dp.DP method), 146
get_config_dict() (faucet.valve.Valve method), 168
get_config_for_api() (in module faucet.config_parser), 143
get_egress_metadata() (in module faucet.faucet_metadata), 153
get_entry() (faucet.valve_table.ValveGroupTable method), 199
get_in_port_match() (faucet.acl.ACL method), 141
get_lacp_dpid_nomination() (faucet.valve.Valve method), 168
get_lacp_flags() (faucet.port.Port method), 160
get_logger() (in module faucet.config_parser_util), 144
get_logger() (in module faucet.valve_util), 200
get_mac_str() (in module faucet.faucet_dot1x), 151
get_meters() (faucet.acl.ACL method), 141
get_mirror_destinations() (faucet.acl.ACL method), 141
get_native_vlan() (faucet.dp.DP method), 146
genode_link_data() (faucet.dp.DP method), 146
get_ports() (faucet.vlan.VLAN method), 203
get_samples() (in module faucet.fctl), 153
get_setting() (faucet.valve_ryuapp.RyuAppBase method), 198
get_setting() (in module faucet.valve.util), 200
genode_link_data() (in module faucet.faucet_util), 200
get_tables() (faucet.dp.DP method), 146
get_tables() (faucet.faucet.Faucet method), 149
get_tables() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
get_tunnel_flowmods() (faucet.valve.Valve method), 169
get_tunnel_id() (faucet.acl.ACL method), 141
gen_tunnel_rule_indices() (faucet.acl.ACL method), 141
global_routing (faucet.valve_route.ValveIPv4RouteManager attribute), 194
global_routing (faucet.valve_route.ValveIPv6RouteManager attribute), 195
global_routing (faucet.valve_route.ValveRouteManager attribute), 196
global_vlan (faucet.valve_route.ValveIPv4RouteManager attribute), 194
global_vlan (faucet.valve_route.ValveIPv6RouteManager attribute), 195
global_vlan (faucet.valve_route.ValveRouteManager attribute), 196
global_vlan (in module faucet.valve.of), 181
goto() (faucet.valve_table.ValveTable method), 199
goto_miss() (faucet.valve_table.ValveTable method), 199
goto_table() (in module faucet.valve.of), 181
goto_this() (faucet.valve_table.ValveTable method),
group_act() (in module faucet.valve_of), 181

host_manager

(group_from_str) (faucet.valve_table.ValveGroupTable static method), 199
groupadd() (in module faucet.valve_of), 181
groupadd_ff() (in module faucet.valve_of), 181
grouppwd() (in module faucet.valve_of), 181
groupdel() (in module faucet.valve_of), 181
groupmod() (in module faucet.valve_of), 181
groupmod_ff() (in module faucet.valve_of), 181
GROUPS (faucet.valve.Valve attribute), 167

H

hairpin_ports() (faucet.valn.VLAN method), 203
healthy_stack_roots()

(faucet.valves_manager.ValvesManager method), 201
host_manager (faucet.valve.AlliedTelesis attribute), 164
host_manager (faucet.valve.ArubaValve attribute), 164
host_manager (faucet.valve.CiscoC9KValve attribute), 165
host_manager (faucet.valve.NoviFlowValve attribute), 165
host_manager (faucet.valve.OVSTfmValve attribute), 166
host_manager (faucet.valve.OVSValve attribute), 166
host_manager (faucet.valve.TfmValve attribute), 167
host_manager (faucet.valve.Valve attribute), 169
HostCacheEntry (class in faucet.vlan), 201
hosts() (faucet.port.Port method), 160
hosts_count() (faucet.port.Port method), 160
hosts_count() (faucet.valn.VLAN method), 203

I

ICMP_SIZE (faucet.valve_route.ValveIPv4RouteManager attribute), 193
ICMP_SIZE (faucet.valve_route.ValveIPv6RouteManager attribute), 194
ICMP_SIZE (faucet.valve_route.ValveRouteManager attribute), 195
ICMP_TYPE (faucet.valve_route.ValveIPv4RouteManager attribute), 194
ICMP_TYPE (faucet.valve_route.ValveIPv6RouteManager attribute), 194
ICMP_TYPE (faucet.valve_route.ValveRouteManager attribute), 195
icmpv6_echo_reply() (in module faucet.valve_packet), 188
ignore_port() (in module faucet.valve_of), 181
ignore_subconf() (faucet.conf.Conf method), 143
in_port_tables() (faucet.dp.DP method), 146

inc_var() (faucet.faucet_metrics.FaucetMetrics method), 153
InfluxShipper (class in faucet.gauge_influx), 157
info() (faucet.valve.ValveLogger method), 173
init_table() (in module faucet.tfm_pipeline), 163
initialise_tables()

(faucet.valve_acl.ValveAclManager method), 174
(faucet.valve_flood.ValveFloodManager method), 175
initialise_tables()

(faucet.valve_manager_base.ValveManagerBase method), 179
initialise_tables()

(faucet.valve_pipeline.ValvePipeline method), 192
int_from_mac (in module faucet.valve_packet), 188
int_in_mac (in module faucet.valve_packet), 188
InvalidConfigError, 143
ip_dsts_for_ip_gw() (faucet.valn.VLAN method), 203
ip_in_vip_subnet() (faucet.valn.VLAN method), 203
IP_PKT (faucet.valve_route.ValveIPv4RouteManager attribute), 194
IP_PKT (faucet.valve_route.ValveIPv6RouteManager attribute), 194
IP_PKT (faucet.valve_route.ValveRouteManager attribute), 195
ip_ver() (faucet.valve_packet.PacketMeta method), 186
ipaddress_fields (faucet.router.Router attribute), 163
IPv (faucet.valve_route.ValveIPv4RouteManager attribute), 194
IPv (faucet.valve_route.ValveIPv6RouteManager attribute), 194
IPv (faucet.valve_route.ValveRouteManager attribute), 195
ipv4_parseable (in module faucet.valve_packet), 188
ipv6_link_eth_mcast (in module faucet.valve_packet), 188
ipv6_solicited_node_from_ucast (in module faucet.valve_packet), 188
ipvs() (faucet.valn.VLAN method), 203
is_active() (faucet.gauge_pollers.GaugePoller static method), 157
is_active() (faucet.gauge_pollers.GaugeThreadPoller method), 158
is_actor_init() (faucet.port.Port method), 160
is_actor_noact() (faucet.port.Port method), 160
is_actor_none() (faucet.port.Port method), 160
is_actor_up() (faucet.port.Port method), 160
is_apply_actions() (in module faucet.valve.of), 181
is_faucet_vip() (faucet.vlan.VLAN method), 203
is_flowdel() (in module faucet.valve.of), 182
is_flowmod() (in module faucet.valve.of), 182
is_global_flowdel() (in module faucet.valve.of), 182
is_global_groupdel() (in module faucet.valve.of), 182
is_groupadd() (in module faucet.valve.of), 182
is_groupdel() (in module faucet.valve.of), 182
is_groupmod() (in module faucet.valve.of), 182
is_host_fib_route() (faucet.vlan.VLAN method), 203
is_in_path() (faucet.dp.DP method), 146
is_meter() (in module faucet.valve.of), 182
is_meteradd() (in module faucet.valve.of), 182
is_meterdel() (in module faucet.valve.of), 182
is_metermod() (in module faucet.valve.of), 183
is_output() (in module faucet.valve.of), 183
is_packetout() (in module faucet.valve.of), 183
is_port_selected() (faucet.port.Port method), 160
is_port_standby() (faucet.port.Port method), 161
is_port_unselected() (faucet.port.Port method), 161
is_registered() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
is_set_field() (in module faucet.valve.of), 183
is_stack_admin_down() (faucet.port.Port method), 161
is_stack_bad() (faucet.port.Port method), 161
is_stack_edge() (faucet.dp.DP method), 146
is_stack_gone() (faucet.port.Port method), 161
is_stack_init() (faucet.port.Port method), 161
is_stack_none() (faucet.dp.DP method), 161
is_stack_root() (faucet.dp.DP method), 146
is_stack_root_candidate() (faucet.dp.DP method), 146
is_stack_up() (faucet.port.Port method), 161
is_table_features_req() (in module faucet.valve.of), 183
is_transit_stack_switch() (faucet.dp.DP method), 146

K
kill_on_exception() (in module faucet.valve_util), 200

L
13_dst (faucet.valve_packet.PacketMeta attribute), 186
13_src (faucet.valve_packet.PacketMeta attribute), 186
13 pkt (faucet.valve_packet.PacketMeta attribute), 186
13_pkt (faucet.valve_packet.PacketMeta attribute), 186
13 Src (faucet.valve_packet.PacketMeta attribute), 186
13_src (faucet.valve_packet.PacketMeta attribute), 186
lacp_actor_up() (in module faucet.valve.packet), 188
lacp_disable_forwarding() (faucet.valve.Valve method), 169
lacp_enable_forwarding() (faucet.valve.Valve method), 169
lacp_handler() (faucet.valve.Valve method), 169
lacp_port_state() (faucet.port.Port method), 161
lacp_ports() (faucet.dp.DP method), 146
lacp_up() (faucet.vlan.VLAN method), 203
lacp_up_port() (faucet.dp.DP method), 146
lacp_up_selected_ports() (faucet.vlan.VLAN method), 203
lacp_update() (faucet.port.Port method), 161
last_retry_time (faucet.valve_route.NextHop attribute), 193
learn_host() (faucet.valve.Valve method), 169
learn_host_intervlan_routing_flows() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_port_flows() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_flow_removed_flows() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_flow_removed_manager() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_port_flows() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_ports() (faucet.valve_host.ValveHostManager method), 179
learn_host_on_vlan_host_flow_removed_manager() (faucet.valve_host.ValveHostManager method), 178
learn_host_on_vlan_host_flow_removed_manager() (faucet.valve_host.ValveHostManager method), 179
link_and_other_vips() (faucet.vlan.VLAN method), 203
lldp_beacon() (in module faucet.valve.packet), 189
lldp_beacon_defaults_types (faucet.dp.DP attribute), 146
lldp_beacon_defaults_types (faucet.port.Port attribute), 161
lldp_beacon_enabled() (faucet.port.Port method), 161
lldp_beacon_send_ports() (faucet.dp.DP method), 146
Faucet Documentation

lldp_handler () (faucet.valve.Valve method), 170
lldp_org_tlv_defaults_types
  (faucet.port.Port attribute), 161
load_configs () (faucet.valves_manager.ValvesManager
  method), 201
load_tables () (in module faucet.tfm_pipeline), 163
log () (faucet.valve_packet.PacketMeta method), 186
log_auth_event () (faucet.faucet_dot1x.FaucetDot1x
  method), 151
log_port_event () (faucet.faucet_dot1x.FaucetDot1x
  method), 151
logger (faucet.gauge_influx.InfluxShipper attribute), 157
logger (faucet.valve.AlliedTelesis attribute), 164
logger (faucet.valve.ArubaValve attribute), 164
logger (faucet.valve.CiscoC9KValve attribute), 165
logger (faucet.valve.NoviFlowValve attribute), 165
logger (faucet.valve.OVSTfmValve attribute), 166
logger (faucet.valve.OVSValve attribute), 166
logger (faucet.valve.TfmValve attribute), 167
logger (faucet.valve.RouteManager attribute), 170
logger (faucet.valve_route.ValveIPv4RouteManager
  attribute), 194
logger (faucet.valve_route.ValveIPv6RouteManager
  attribute), 195
logname (faucet.faucet.Faucet attribute), 149
logname (faucet.gauge.Gauge attribute), 154
logname (faucet.valve.AlliedTelesis attribute), 164
logname (faucet.valve.ArubaValve attribute), 164
logname (faucet.valve.CiscoC9KValve attribute), 165
logname (faucet.valve.NoviFlowValve attribute), 165
logname (faucet.valve.OVSTfmValve attribute), 166
logname (faucet.valve.OVSValve attribute), 166
logname (faucet.valve.RouteManager attribute), 170
logname (faucet.valve.ryuapp.RyuAppBase attribute), 198
logoff_handler () (faucet.faucet_dot1x.FaucetDot1x
  method), 151
loop_protect_external_ports ()
  (faucet.vlan.VLAN method), 204
loop_protect_external_ports_up ()
  (faucet.vlan.VLAN method), 204

M

mac_addr_all_zeros (in module faucet.valve_packet), 190
mac_addr_is_unicast (in module faucet.valve_packet), 190
mac_byte_mask () (in module faucet.valve_packet), 190
mac_mask_bits (in module faucet.valve_packet), 190
main() (in module faucet.check_faucet_config), 142
maintain_stack_root()
  (faucet.valves_manager.ValvesManager
    method), 201
make_point () (faucet.gauge_influx.InfluxShipper
  static method), 157
make_port_point ()
  (faucet.gauge_influx.InfluxShipper method), 157
make_wsgi_app() (in module faucet.prom_client), 162
match (faucet.valve_table.ValveTable attribute), 199
match () (in module faucet.valve.of), 183
match_from_dict () (in module faucet.valve.of), 183
match_tables () (faucet.dp.DP method), 146
MAX_ETH_TYPE_PKT_SIZE
  (faucet.valve_packet.DP attribute), 183
max_host_fib_retry_count
  (faucet.valve_route.ValveIPv4RouteManager
    attribute), 194
max_host_fib_retry_count
  (faucet.valve_route.ValveIPv6RouteManager
    attribute), 195
max_host_fib_retry_count
  (faucet.valve_route.ValveRouteManager
    attribute), 196
max_hosts_per_resolve_cycle
  (faucet.valve_route.ValveIPv4RouteManager
    attribute), 194
max_hosts_per_resolve_cycle
  (faucet.valve_route.ValveIPv6RouteManager
    attribute), 195
max_hosts_per_resolve_cycle
  (faucet.valve_route.ValveRouteManager
    attribute), 196
MAX_PACKET_IN_SIZE
  (faucet.valve_route.ValveRouteManager
    attribute), 195
max_resolve_backoff_time
  (faucet.valve_route.ValveIPv4RouteManager
    attribute), 194
max_resolve_backoff_time
  (faucet.valve_route.ValveIPv6RouteManager
    attribute), 195
max_resolve_backoff_time
  (faucet.valve_route.ValveRouteManager
    attribute), 196
MAX_TABLE_ID
  (faucet.valve.OVSTfmValve attribute), 165
MAX_TABLE_ID
  (faucet.valve.TfmValve attribute), 166
merge_dyn () (faucet.conf.Conf method), 143
metadata_goto_table()  (in module faucet.valve_of), 183
MetaDPState (class in faucet.valves_manager), 200
Meter (class in faucet.meter), 159
meter_id (faucet.meter.Meter attribute), 159
meteradd() (in module faucet.valve_of), 184
meterdel() (in module faucet.valve_of), 184
metric_update() (faucet.faucet.Faucet method), 149
metrics (faucet.valve.AlliedTelesis attribute), 164
metrics (faucet.valve.ArubaValve attribute), 164
metrics (faucet.valve.CiscoC9KValve attribute), 165
metrics (faucet.valve.NoviFlowValve attribute), 165
metrics (faucet.valve.OVSTfmValve attribute), 166
metrics (faucet.valve.OVSValve attribute), 166
metrics (faucet.valve.TfmValve attribute), 167
metrics (faucet.valve bras.Valve attribute), 170
MIN_ETH_TYPE_PKT_SIZE  (faucet.valve_packet.PacketMeta attribute), 186
MIN_MAX_FLOWS  (faucet.valve.OVSTfmValve attribute), 165
MIN_MAX_FLOWS  (faucet.valve.TfmValve attribute), 166
mirror_actions() (faucet.port.Port method), 161
mirrored_ports() (faucet.vlan.VLAN method), 204
modify() (faucet.valve_table.ValveGroupEntry method), 199
modify_stack_topology() (faucet.dp.DP static method), 147
multi_out() (faucet.valve_route.ValveIPv4RouteManager attribute), 194
multi_out() (faucet.valve_route.ValveIPv6RouteManager attribute), 195
multi_out() (faucet.valve_route.ValveRouteManager attribute), 196
mutableattrs (faucet.conf.Conf attribute), 143
mutableattrs (faucet.dp.DP attribute), 147
mutableattrs (faucet.vlan.VLAN attribute), 204
N
name (faucet.vlan.AnyVLAN attribute), 201
name (faucet.vlan NullVLAN attribute), 202
nd_advert (in module faucet.valve_packet), 190
nd_request (in module faucet.valve_packet), 190
neigh_cache_by_ip() (faucet.vlan.VLAN method), 204
neigh_cache_count_by_ip() (faucet.vlan.VLAN method), 204
neighbor_timeout() (faucet.valve_route.ValveIPv4RouteManager attribute), 194
neighbor_timeout() (faucet.valve_route.ValveIPv6RouteManager attribute), 195
neighbor_timeout() (faucet.valve_route.ValveRouteManager attribute), 197
new_value() (faucet.valves_manager.ValvesManager method), 201
next_retry() (faucet.valve_route.NextHop method), 193
next_retry_time() (faucet.valve_route.NextHop attribute), 193
NextHop (class in faucet.valve.route), 193
nexthop_dead() (faucet.valve_route.ValveRouteManager method), 197
nfv_sw_port_up() (faucet.faucet_dot1x.FaucetDot1x method), 151
no_response() (faucet.gauge_pollers.GaugePoller method), 157
no_response() (faucet.gauge_pollers.GaugePortStatePoller method), 158
no_response() (faucet.watchers.GaugePortStatePoller static method), 205
non_stack_forwarding() (faucet.port.Port method), 161
non_vlan_ports() (faucet.dp.DP method), 147
NonBlockLock (class in faucet.faucet_event), 152
notifier (faucet.faucet.Faucet attribute), 149
notifier (faucet.valve.AlliedTelesis attribute), 164
notifier (faucet.valve.ArubaValve attribute), 164
notifier (faucet.valve.CiscoC9KValve attribute), 165
notifier (faucet.valve.NoviFlowValve attribute), 165
notifier (faucet.valve.OVSValve attribute), 166
notifier (faucet.valve.OVSTfmValve attribute), 166
notifier (faucet.valve.TfmValve attribute), 167
notifier (faucet.valve bras.Valve attribute), 170
notify() (faucet.valve_route.ValveIPv4RouteManager attribute), 194
notify() (faucet.valve_route.ValveIPv6RouteManager attribute), 195
notify() (faucet.valve_route.ValveRouteManager attribute), 197
notify() (faucet.faucet_event.FaucetEventNotifier method), 152
notify() (faucet.valve.Valve method), 170
notify_learn() (faucet.valve_route.ValveRouteManager method), 197
NoviFlowValve (class in faucet.valve), 165
NullVLAN (class in faucet.vlan), 202
O
ofchannel_log() (faucet.valve.Valve method), 170
ofchannel_logger (faucet.valve.AlliedTelesis attribute), 164
ofchannel_logger (faucet.valve.ArubaValve attribute), 164
ofchannel_logger (faucet.valve.CiscoC9KValve attribute), 165
ofchannel_logger (faucet.valve bras.Valve attribute), 165
Index 223
ofchannel_logger (faucet.valve.OVSTfmValve attribute), 166
ofchannel_logger (faucet.valve.OVSValve attribute), 166
ofchannel_logger (faucet.valve.TfmValve attribute), 167
ofchannel_logger (faucet.valve.Valve attribute), 170
ofdescstats_handler() (faucet.valve.Valve attribute), 170
oferror() (faucet.valve.Valve method), 170
OFPP_VERSIONS (faucet.valve.ryuapp.RyuAppBase attribute), 198
OFVLAN (class in faucet.vlan), 202
orig_len (faucet.valve.packet.PacketMeta attribute), 186
output() (faucet.valve.pipeline.ValvePipeline method), 192
output_actions_types (faucet.acl.ACL attribute), 141
output_controller (in module faucet.valve_of), 184
output_in_port (in module faucet.valve_of), 184
output_non_output_actions (in module faucet.valve_of), 184
output_port (in module faucet.valve_of), 184
output_table() (faucet.dp.DP method), 147
output_tables() (faucet.dp.DP method), 147
OVSTfmValve (class in faucet.valve), 165
OVSValve (class in faucet.valve), 166

PacketMeta (class in faucet.valve.packet), 186
packet_in_handler() (faucet.faucet.Faucet method), 149
PacketMeta (class in faucet.valve.packet), 186
packetout() (in module faucet.valve_of), 184
packetouts() (in module faucet.valve_of), 184
parse_args() (in module faucet.fctf), 154
parse_configs() (faucet.valves_manager.ValvesManager method), 201
parse_eth_pkt() (in module faucet.valve.packet), 190
parse_faucet_lldp() (in module faucet.valve.packet), 191
parse_lacp_pkt() (in module faucet.valve.packet), 191
parse_lldp() (in module faucet.valve.packet), 191
parse_packet_in_pkt (in module faucet.valve.packet), 191
parse_pkt_meta() (faucet.valve.Valve method), 170
parse_rcv_packet() (faucet.valve.Valve method), 170
peer_stack_up_ports() (faucet.dp.DP method), 147
pipeline (faucet.valve.AlliedTelesis attribute), 164
pipeline (faucet.valve.ArubaValve attribute), 164
pipeline (faucet.valve.CiscoC9KValve attribute), 165
pipeline (faucet.valve.NoviflowValve attribute), 165
pipeline (faucet.valve.OVSTfmValve attribute), 166
pipeline (faucet.valve.OVSValve attribute), 166
pipeline (faucet.valve.TfmValve attribute), 167
pipeline (faucet.valve.Valve attribute), 170
pipeline (faucet.valve_route.ValveIPv4RouteManager attribute), 194
pipeline (faucet.valve_route.ValveIPv6RouteManager attribute), 195
pkt (faucet.valve.packet.PacketMeta attribute), 186
pkt_out_port() (faucet.vlan.VLAN method), 204
pop_vlan (in module faucet.valve_of), 185
Port (class in faucet.port), 160
port (faucet.valve.packet.PacketMeta attribute), 186
port (faucet.valve_route.NextHop attribute), 193
port (faucet.vlan.HostCacheEntry attribute), 202
port_add() (faucet.valve.Valve method), 170
port_delete() (faucet.valve.Valve method), 171
port_down() (faucet.faucet_dot1x.FaucetDot1x method), 151
port_is_tagged() (faucet.vlan.VLAN method), 204
port_is_untagged() (faucet.vlan.VLAN method), 204
port_labels() (faucet.dp.DP method), 147
port_no_valid() (faucet.dp.DP method), 147
port_role_name() (faucet.port.Port method), 161
port_status_from_state() (in module faucet.valve_of), 185
port_status_handler() (faucet.faucet.Faucet method), 149
port_status_handler() (faucet.valve.Valve method), 171
port_status_handler() (faucet.valves_manager.ValvesManager method), 201
port_up() (faucet.faucet_dot1x.FaucetDot1x method), 151
ports_add() (faucet.valve.Valve method), 171
ports_delete() (faucet.valve.Valve method), 171
ports_from_output_portActs() (in module faucet.valve_of), 185
prepare_send_flows() (faucet.valve.Valve method), 171
proactive_learn(faucet.valve_route.ValveIPv4RouteManager attribute), 194
proactive_learn(faucet.valve_route.ValveRouteManager, 195)
proactive_learn(faucet.valve_route.ValveRouteManager.port_dp_status(), 197)
PromClient(class in faucet.prom_client, 162)
push_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 152
push_vlan() (in module faucet.valve_acl), 174
push_vlan_act (in module faucet.valve_of), 185
rate_limit_packet_ins() (faucet.valve.Valve method), 171
rcv_packet() (faucet.valve.Valve method), 171
read_config() (in module faucet.config_parser_util), 144
recent_ofmsgs (faucet.valve.AlliedTelesis attribute), 164
recent_ofmsgs (faucet.valve.ArubaValve attribute), 164
recent_ofmsgs (faucet.valve.CiscoC9KValve attribute), 165
recent_ofmsgs (faucet.valve.NoviFlowValve attribute), 165
recent_ofmsgs (faucet.valve.OVSTfmValve attribute), 166
recent_ofmsgs (faucet.valve.OVSValve attribute), 166
recent_ofmsgs (faucet.valve.TfmValve attribute), 167
recent_ofmsgs (faucet.valve.Valve attribute), 171
reconnect_handler() (faucet.valve_ryuapp.RyuAppBase method), 198
release() (faucet.faucet_event.NonBlockLock method), 152
reload_config() (faucet.faucet.Faucet method), 149
reload_config() (faucet.faucet_experimental_api.FaucetExperimentalAPI method), 153
reload_config() (faucet.gauge.Gauge method), 154
reload_config() (faucet.valve.Valve method), 171
reload_config() (faucet.valve_ryuapp.RyuAppBase method), 198
remove_filter() (faucet.valve_pipeline.ValvePipeline method), 193
remove_non_tunnel_rules() (faucet.acl.ACL method), 142
remove_stack_link() (faucet.dp.DP class method), 147
reparse() (faucet.valve_packet.PacketMeta method), 186
reparse_all() (faucet.valve_packet.PacketMeta method), 186
rcv_msg() (faucet.valve yönlendirici.ValeRouteManager, 195)
rcv_msg() (faucet.valve yönlendirici.ValveRouteManager.port_dp_status(), 197)
required_labels (faucet.prom_client.PromClient attribute), 162
reregister_flow_vars() (faucet.gauge_prom.GaugePrometheusClient method), 159
reset() (faucet.faucet_bgp.FaucetBgp method), 150
reset() (faucet.faucet_dot1x.FaucetDot1x method), 151
reset_caches() (faucet.valve_vlan.VLAN method), 204
reset_dpid() (faucet.valve_metrics.FaucetMetrics method), 153
reset_ports() (faucet.valve_vlan.VLAN method), 204
reset_refs() (faucet.dp.DP method), 147
resolution_due() (faucet.valve_routing.NextHop method), 193
resolve_expired_hosts() (faucet.valve_routing.ValveRouteManager method), 197
resolve_gateways() (faucet.valve.Valve method), 172
resolve_gateways() (faucet.valve_routing.ValveRouteManager method), 197
resolve_gateways() (faucet.valve_routing.ValveRouteManager method), 197
resolve_stack_topology() (faucet.dp.DP method), 147
restricted_bcast_arpnd_ports() (faucet.dp.DP method), 147
restricted_bcast_arpnd_ports() (faucet.valve_vlan.VLAN method), 204
restricted_flood_dists (faucet.valve_flood.ValveFloodManager attribute), 175
revert_config() (faucet.valves_manager.ValvesManager method), 201
rewrite_vlan() (in module faucet.valve_vlan), 175
route_count_by_ipv() (faucet.valve_vlan.VLAN method), 204
route_priority (faucet.valve_routing.ValveIPv4RouteManager attribute), 194
route_priority (faucet.valve_routing.ValveIPv6RouteManager attribute), 194
method), 158
start() (faucet.prom_client.PromClient method), 162
start() (faucet.valve_ryuapp.RyuAppBase method), 198
stat_config_files() (in module faucet.valve_util), 200
state_expire() (faucet.valve.Valve method), 172
STATIC_TABLE_IDS (faucet.valve.NoviFlowValve attribute), 165
STATIC_TABLE_IDS (faucet.valve.Valve attribute), 167
stop() (faucet.gauge_pollers.GaugePoller method), 158
stop() (faucet.gauge_pollers.GaugeThreadPoller method), 158
switch_features() (faucet.valve.Valve method), 173
table_by_id() (faucet.dp.DP method), 147
table_features() (in module faucet.valve_of), 185
tagged_flood_ports() (faucet.vlan.VLAN method), 204
test_config_condition() (in module faucet.conf), 143
TfmValve (class in faucet.valve), 166
tlv_cast() (in module faucet.valve_packet), 191
tlvs_by_subtype() (in module faucet.valve_packet), 191
tlvs_by_type() (in module faucet.valve_packet), 191
to_conf() (faucet.conf.Conf method), 143
tunnel_types (faucet.acl.ACL attribute), 142

UniqueKeyLoader (class in faucet.config_parser_util), 144
unpack_tunnel() (faucet.acl.ACL method), 142
untagged_flood_ports() (faucet.vlan.VLAN method), 204
update() (faucet.conf.Conf method), 143
update() (faucet.gauge_pollers.GaugePoller method), 158
update() (faucet.valves_manager.ConfigWatcher method), 200
update_buckets() (faucet.valve_table.ValveGroupEntry method), 199
update_config_applied() (faucet.valves_manager.ValvesManager method), 201
update_config_metrics() (faucet.valve.Valve method), 173
update_metrics() (faucet.valve.Valve method), 173
update_metrics() (faucet.valves_manager.ValvesManager method), 201
update_stack_topo() (faucet.valve_flood.ValveFloodManager static method), 175
update_stack_topo() (faucet.valve_flood.ValveFloodStackManagerBase method), 176
update_tunnel_acl_conf() (faucet.acl.ACL method), 142
update_tunnel_flowrules() (faucet.valve.Valve method), 173
update_vlan() (faucet.valve_flood.ValveFloodManager method), 175
update_vlan() (faucet.valves_manager_base.ValveManagerBase method), 179
update_watcher_handler() (faucet.gauge.Gauge method), 154
USE_BARRIERS (faucet.valve.NoviFlowValve attribute), 165
USE_BARRIERS (faucet.valve.OVSTfmValve attribute), 165
USE_BARRIERS (faucet.valve.OVSValve attribute), 167
USE_OXM_IDS (faucet.valve.OVSTfmValve attribute), 166
USE_OXM_IDS (faucet.valve.TfmValve attribute), 166
utf8_decode() (in module faucet.valve_util), 200

Valve (class in faucet.valve), 167
valve_factory() (in module faucet.valve), 173
valve_flow_services() (faucet.valves_manager.ValvesManager method), 201
valve_flowreorder() (in module faucet.valve_of), 185
valve_match_vid (in module faucet.valve_of), 185
valve_packet_in() (faucet.valves_manager.ValvesManager method), 201
ValveAclManager (class in faucet.valve_acl), 173
ValveDeadThreadException, 198
ValveFloodManager (class in faucet.valve_flood), 175
ValveFloodManagerBase (class in faucet.valve_flood), 175
ValveFloodStackManagerNoReflection (class in faucet.valve_flood), 176
ValveFloodStackManagerReflection (class in faucet.valve_flood), 176
ValveGroupEntry (class in faucet.valve_table), 198
ValveGroupTable (class in faucet.valve_table), 199
ValveHostFlowRemovedManager (class in faucet.valve_host), 178
ValveHostManager (class in faucet.valve_host), 178
ValveIPv4RouteManager (class in faucet.valve_route), 193
ValveIPv6RouteManager (class in faucet.valve_route), 194
ValveLogger (class in faucet.valve), 173
ValveManagerBase (class in faucet.valve_manager_base), 179
ValvePipeline (class in faucet.valve_pipeline), 192
ValveRouteManager (class in faucet.valve_route), 195
valves (faucet.valves_manager.ValvesManager attribute), 201
valves_manager (faucet.faucet.Faucet attribute), 149
ValvesManager (class in faucet.valves_manager), 200
ValveTable (class in faucet.valve_table), 199
ValveTableConfig (class in faucet.faucet_pipeline), 153
verify_tunnel_rules() (faucet.acl.ACL method), 142
vid (faucet.vlan.AnyVLAN attribute), 201
vid (faucet.vlan.NullVLAN attribute), 202
vid_present() (in module faucet.valve_of), 185
vid_valid() (faucet.vlan.VLAN static method), 204
vip_map() (faucet.router.Router method), 163
vip_map() (faucet.vlan.VLAN method), 204
vip_table (faucet.valve_route.ValveIPv4RouteManager attribute), 194
vip_table (faucet.valve_route.ValveIPv6RouteManager attribute), 195
vip_table (faucet.valve_route.ValveRouteManager attribute), 197
VLAN (class in faucet.vlan), 202
vlan (faucet.valve_packet.PacketMeta attribute), 186
vlan_pkt (faucet.valve_packet.PacketMeta attribute), 186
vlans() (faucet.port.Port method), 162

W
warning() (faucet.valve.ValveLogger method), 173
watcher_factory() (in module faucet.watcher), 205
watcher_parser() (in module faucet.config_parser), 143
WatcherConf (class in faucet.watcher_conf), 205