nmeta Documentation

Release 0.4.3
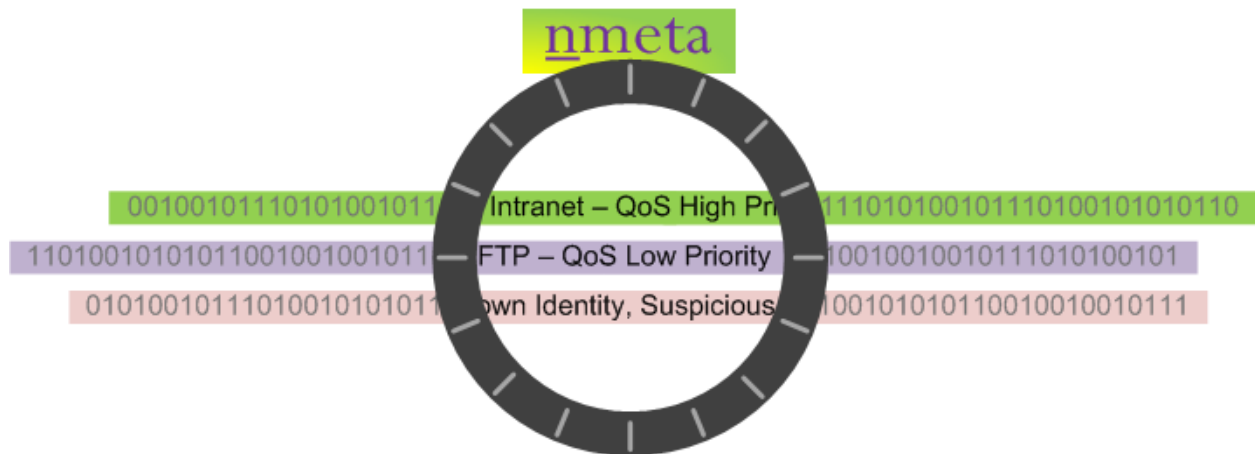
Matthew John Hayes

Oct 28, 2017

Contents

The nmeta project is a research platform for traffic classification on Software Defined Networking (SDN). Read More

Contents:

Introduction

The nmeta *(pronounced en-meta)* project is founded on the belief that innovation in networks requires a foundation layer of knowledge about both the participants and their types of conversation.

Today, networks generally have only a limited view of participants and conversation types
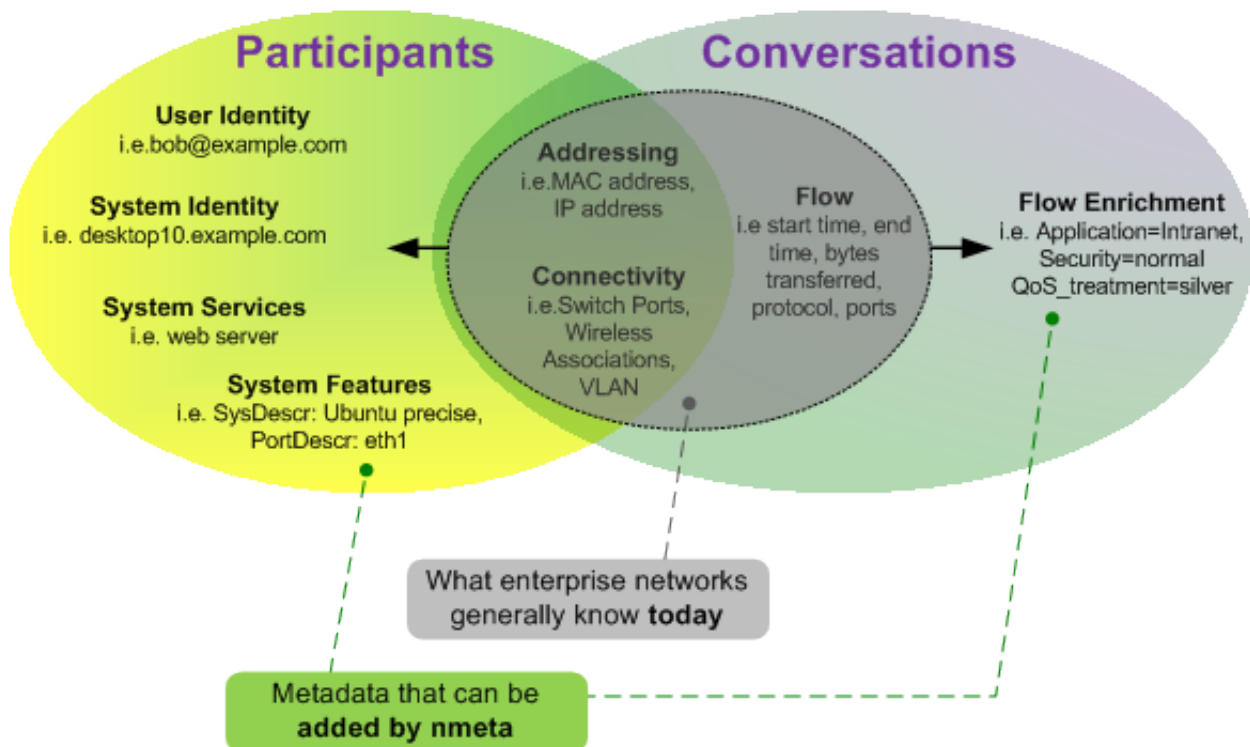
The goal of the nmeta project is to produce network metadata enriched with participant identities and conversation types to provide a foundation for innovation in networking.

The production of enriched network metadata requires policy-based control, and ability to adapt to new purposes

through extensibility.

Enriched network metadata has a number of uses, including classifying flows for QoS, billing, traffic engineering, troubleshooting and security.

Nmeta is a research platform for traffic classification on Software Defined Networking (SDN). It runs on top of the Ryu SDN controller (see: http://osrg.github.io/ryu/).

How it Works

Nmeta uses OpenFlow Software-Defined Networking (SDN) to selectively control flows through switches so that packets can be classified and actions taken. It instructs connected OpenFlow switches to send packets from unknown flows to the Ryu SDN Controller, on which nmeta runs, for analysis.
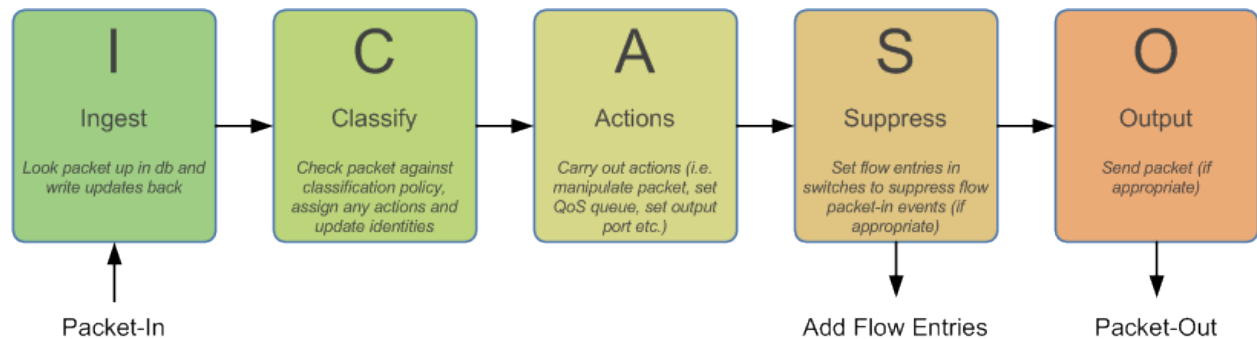
Nmeta configures a single flow table per switch with a table-miss flow entry (FE) that sends full unmatched packets to the controller. As flows are classified, specific higher-priority FEs are configured to suppress sending further packets to the controller.

Limitations

Nmeta does not scale well. Every new flow has a processing overhead, and this workload cannot be scaled horizontally on the controller. The nmeta2 system is being developed to address this limitation.

Nmeta has no security, it was written to demonstrate SDN functionality and has omitted addressing security requirements. A future rewrite may address security, but for now there is no security whatsoever.

Feature Enhancement Wishlist

See Issues for list of enhancements and bugs

Privacy Considerations

Collecting network metadata brings with it ethical and legal considerations around privacy. Please ensure that you have permission to monitor traffic before deploying this software.

Disclaimer

This code carries no warrantee whatsoever. Use at your own risk.

How to Contribute

Code contributions and suggestions are welcome. Enhancement or bug fixes can be raised as issues through GitHub.

Please get in touch if you want to be added as a contributor to the project:

Email: Nmeta Maintainer

Recipes

These recipes are to provide ideas on how nmeta can be used through examples.

Note that policies have an **implicit allow** at the end of the policy. Also, actions implicitly allow if there is no drop action.

Recipes:

- *Parental Control Recipe*
- *LAN Traffic Clean-up*
- *Quality of Service (QoS) Recipe*
- *ML Training Data Collector*

Parental Control Recipe

This recipe is for using nmeta to provide parental control on a home network. It is just an example of some capabilities, the exact configuration needs to be tailored to your specific requirements. Note that parental controls on network should be part of a wider strategy, including controls on the devices used by children, and education on internet safety.

In this fictional example, there are two children, conveniently named Alice and Bob. Alice has a Chromebook, which does not register a hostname via DHCP, but does have a consistent Wi-Fi MAC address (01:23:45:67:89:ab). Bob has an iPhone with a DHCP host name of *Bobs-iPhone*.

In this recipe we enforce the following parental controls on Alice and Bob:

- All devices on the home network can only do DNS lookups against OpenDNS FamilyShield servers (that attempt to block adult content), apart from Chromecast which doesn't honour the DNS allocations in DHCP and insists on talking to Google's DNS servers

- Alice's Chromebook is blocked from accessing YouTube

- Alice's Chromebook and Bob's iPhone are only allowed to access the Internet between 7am and 9pm

Main Policy

Use this main_policy.yaml file in the user config directory:

```
~/nmeta/nmeta/config/user/
```

Here's the YAML:

```yaml
---
#*** Main Policy for nmeta - Home Router Parental Control Policy
#*** Written in YAML
#
tc_rules:
    # Traffic Classification Rulesets and Rules
    tc_ruleset_1:
        - comment: Allow OpenDNS
          match_type: any
          conditions_list:
              - match_type: all
                classifiers_list:
                    - udp_dst: 53
                    - ip_dst: 208.67.222.123
              - match_type: all
                classifiers_list:
                    - udp_src: 53
                    - ip_src: 208.67.222.123
              - match_type: all
                classifiers_list:
                    - udp_dst: 53
                    - ip_dst: 208.67.220.123
              - match_type: all
                classifiers_list:
                    - udp_src: 53
                    - ip_src: 208.67.220.123
          actions:
              set_desc: "OpenDNS Name Resolution"
        - comment: Allow Chromecast DNS to Google
          match_type: any
          conditions_list:
              - match_type: all
                classifiers_list:
                    - identity_dhcp_hostname: Chromecast
                    - udp_dst: 53
                    - ip_dst: 8.8.8.8
              - match_type: all
                classifiers_list:
                    - identity_dhcp_hostname: Chromecast
                    - udp_src: 53
                    - ip_src: 8.8.8.8
          actions:
              set_desc: "Allow Chromecast DNS to Google"
        - comment: Block all other DNS
          match_type: any
          conditions_list:
              - match_type: any
                classifiers_list:
                    - udp_src: 53
              - match_type: any
```

```
                    classifiers_list:
                        - udp_dst: 53
                - match_type: any
                  classifiers_list:
                        - tcp_src: 53
                - match_type: any
                  classifiers_list:
                        - tcp_dst: 53
            actions:
                set_desc: "Bad DNS, needs investigating"
                drop: at_controller
        - comment: Drop Alice Chromebook to YouTube
          match_type: any
          conditions_list:
                - match_type: all
                  classifiers_list:
                        - eth_src: 01:23:45:67:89:ab
                        - identity_service_dns_re: '.*\.youtube\*'
                - match_type: all
                  classifiers_list:
                        - eth_src: 01:23:45:67:89:ab
                        - identity_service_dns_re: '.*\.googlevideo\.com'
            actions:
                set_desc: "Drop Alice Chromebook to YouTube"
                drop: at_controller
        - comment: Time of Day restriction on Alice and Bob
          match_type: all
          conditions_list:
                - match_type: any
                  classifiers_list:
                        - eth_src: 01:23:45:67:89:ab
                        - identity_dhcp_hostname: Bobs-iPhone
                - match_type: all
                  classifiers_list:
                        - time_of_day: 21:00-06:59
            actions:
                set_desc: "Drop Kids Internet after hours"
                drop: at_controller
#
qos_treatment:
  # Control Quality of Service (QoS) treatment mapping of
  #  names to output queue numbers:
  default_priority: 0
  constrained_bw: 1
  high_priority: 2
  low_priority: 3
#
port_sets:
    # Port Sets control what data plane ports policies and
    #  features are applied on. Names must be unique.
    port_set_list:
        - name: port_set_location_internal
          port_list:
                - name: TPLink-internal
                  DPID: 1
                  ports: 1-2,4
                  vlan_id: 0
```

```
            - name: port_set_location_external
              port_list:
                  - name: TPLink-external
                    DPID: 1
                    ports: 3
                    vlan_id: 0
#
locations:
    # Locations are logical groupings of ports. Takes first match.
    locations_list:
        - name: internal
          port_set_list:
            - port_set: port_set_location_internal

        - name: external
          port_set_list:
            - port_set: port_set_location_external

    default_match: unknown
```

LAN Traffic Clean-up

This recipe blocks undesirable LAN traffic. What counts as undesirable is up for debate, this recipe just demonstrates some mechanisms for writing a policy

It does the following:

- Drops SSDP (UPnP) traffic

- Drops Bonjour traffic

- Implicit allow of all other traffic, as well of harvesting of conversation and identity metadata

Main Policy

Use this main_policy.yaml file in the user config directory:

```
~/nmeta/nmeta/config/user/
```

Here's the YAML:

```
---
#*** Main Policy for nmeta - Home Router LAN Clean-up Policy
#*** Written in YAML
#
tc_rules:
    # Traffic Classification Rulesets and Rules
    tc_ruleset_1:
        - comment: Drop Bonjour Sleep Proxy
          match_type: any
          conditions_list:
              - match_type: all
                classifiers_list:
                    - udp_src: 5353
                    - udp_dst: 5353
```

```
        actions:
            set_desc: "Drop Bonjour Sleep Proxy"
            drop: at_controller_and_switch
      - comment: Drop SSDP UPnP
        match_type: any
        conditions_list:
            - match_type: all
              classifiers_list:
                  - ip_dst: 239.255.255.250
                  - udp_dst: 1900
        actions:
            set_desc: "Drop SSDP UPnP"
            drop: at_controller_and_switch

#
qos_treatment:
  # Control Quality of Service (QoS) treatment mapping of
  #  names to output queue numbers:
  default_priority: 0
  constrained_bw: 1
  high_priority: 2
  low_priority: 3
#
port_sets:
    # Port Sets control what data plane ports policies and
    #  features are applied on. Names must be unique.
    port_set_list:
        - name: port_set_location_internal
          port_list:
              - name: TPLink-internal
                DPID: 1
                ports: 1-2,4
                vlan_id: 0

        - name: port_set_location_external
          port_list:
              - name: TPLink-external
                DPID: 1
                ports: 3
                vlan_id: 0
#
locations:
    # Locations are logical groupings of ports. Takes first match.
    locations_list:
        - name: internal
          port_set_list:
            - port_set: port_set_location_internal

        - name: external
          port_set_list:
            - port_set: port_set_location_external

    default_match: unknown
```

Quality of Service (QoS) Recipe

This recipe uses QoS to constrain bandwidth of YouTube video traffic, purely as an example of how to do QoS.

Traffic is identified with a classification list, then marked with a QoS treatment action (constrained_bw).

The *qos_treatment* section maps *constrained_bw* to QoS queue number 1.

QoS queues need to be separately configured on switches. Failure to have a queue defined on the switch (other than 0) may result in traffic being dropped.

Main Policy

Use this main_policy.yaml file in the user config directory:

```
~/nmeta/nmeta/config/user/
```

Here's the YAML:

```
---
#*** Main Policy for nmeta - Example QoS Recipe.
#*** Written in YAML
#
# Example QoS constraint of YouTube Video traffic
#
tc_rules:
    # Traffic Classification Rulesets and Rules
    tc_ruleset_1:
        - comment: Constrained Bandwidth Traffic
          match_type: any
          conditions_list:
              - match_type: any
                classifiers_list:
                    - identity_service_dns_re: '.*\.youtube\*'
                    - identity_service_dns_re: '.*\.googlevideo\.com'
          actions:
            set_desc: "Constrained YouTube Bandwidth Traffic"
            qos_treatment: constrained_bw
#
qos_treatment:
    # Control Quality of Service (QoS) treatment mapping of
    #  names to output queue numbers:
    default_priority: 0
    constrained_bw: 1
    high_priority: 2
    low_priority: 3
#
port_sets:
    # Port Sets control what data plane ports policies and
    #  features are applied on. Names must be unique.
    port_set_list:
        - name: port_set_location_internal
          port_list:
              - name: VirtualSwitch1-internal
                DPID: 1
                ports: 1-3,5,66
                vlan_id: 0
```

```
                 - name: VirtualSwitch2-internal
                   DPID: 255
                   ports: 3,5
                   vlan_id: 0

          - name: port_set_location_external
            port_list:
                 - name: VirtualSwitch1-external
                   DPID: 1
                   ports: 6
                   vlan_id: 0

                 - name: VirtualSwitch2-external
                   DPID: 255
                   ports: 1-2,4
                   vlan_id: 0
#
locations:
    # Locations are logical groupings of ports. Takes first match.
    locations_list:
        - name: internal
          port_set_list:
            - port_set: port_set_location_internal

        - name: external
          port_set_list:
            - port_set: port_set_location_external

    default_match: external
```

ML Training Data Collector

This recipe can be used to build traffic classification training data for supervised machine learning (ML). It uses a custom classifier to write flow characteristics into the classification tag. This data can then be retrieved via the classifications API and annotated against the ground truth of what type of flow it was.

Main Policy

Use this main_policy.yaml file in the user config directory:

```
~/nmeta/nmeta/config/user/
```

Here's the YAML:

```
---
#*** Main Policy for nmeta - Machine Learning (ML) Data Collector
#*** Written in YAML
#
tc_rules:
    # Traffic Classification Rulesets and Rules
    tc_ruleset_1:
        - comment: Machine Learning Data Collector
          match_type: any
```

```
                conditions_list:
                    - match_type: any
                      classifiers_list:
                          - custom: ml_training_data_collector_1
            actions:
              set_desc: classifier_return
              qos_treatment: classifier_return
#
qos_treatment:
    # Control Quality of Service (QoS) treatment mapping of
    #  names to output queue numbers:
    default_priority: 0
    constrained_bw: 1
    high_priority: 2
    low_priority: 3
#
port_sets:
    # Port Sets control what data plane ports policies and
    #  features are applied on. Names must be unique.
    port_set_list:
        - name: port_set_location_internal
          port_list:
              - name: VirtualSwitch1-internal
                DPID: 1
                ports: 1-3,5,66
                vlan_id: 0

              - name: VirtualSwitch2-internal
                DPID: 255
                ports: 3,5
                vlan_id: 0

        - name: port_set_location_external
          port_list:
              - name: VirtualSwitch1-external
                DPID: 1
                ports: 6
                vlan_id: 0

              - name: VirtualSwitch2-external
                DPID: 255
                ports: 1-2,4
                vlan_id: 0
#
locations:
    # Locations are logical groupings of ports. Takes first match.
    locations_list:
        - name: internal
          port_set_list:
            - port_set: port_set_location_internal

        - name: external
          port_set_list:
            - port_set: port_set_location_external

    default_match: external
```

Classification data can be retrieved with cURL with this command:

```
curl -g http://localhost:8081/v1/classifications?where={%22classified%22:true} |␣
↪python -m json.tool
```

Example result:

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
            "_etag": "bd0da01a8db1ece3e2f23ecd577bb0474568e268",
            "_id": "59f43d3301186133ec5a8d0f",
            "_updated": "00:00:00.000000",
            "actions": {
                "qos_treatment": "default_priority",
                "set_desc": "classifier_return"
            },
            "classification_tag": "ML,10.1.0.2,10.1.0.1,6,80,35044,162,2.152,0.111,5,
↪[1, 0, 1, 1, 0],[74, 74, 66, 84, 66]",
            "classification_time": "21:17:55.823000",
            "classified": true,
            "flow_hash": "8ac72c304d7c7a61349ba99e0c21541e"
        }
    ],
    "_meta": {
        "max_results": 25,
        "page": 1,
        "total": 1
    }
}
```

Note the classification_tag that contains the flow characteristics. See the custom classifier code for field descriptions.

Install

This guide is for installing on Ubuntu 16.04.2 LTS

Pre-Work

Ensure packages are up-to-date

```
sudo apt-get update
sudo apt-get upgrade
```

Install Debian Packages

The following command installs these packages:

- pip (Python package manager)
- git (version control system)
- git flow (branching model for Git)
- python-pytest (used to run unit tests)
- python-yaml (YAML parser for Python)

```
sudo apt-get install python-pip git git-flow python-pytest python-yaml
```

Install Python Packages

Ensure pip (Python package manager) is latest version:

```
pip install --upgrade pip
```

Nmeta requires Python version 2.7.x, does not run on Python 3.x (yet)

The following command installs these Python packages:

- Ryu (OpenFlow Software-Defined Networking (SDN) controller application that handles communications with the switch)

- dpkt (library is used to parse and build packets)

- mock (Testing library)

- Requests (HTTP library)

- simplejson (JSON encoder and decoder)

- eve (REST API framework)

- coloredlogs (Add colour to log entries in terminal output)

- voluptuous (data validation library)

```
pip2.7 install ryu dpkt mock requests simplejson eve coloredlogs voluptuous --user
```

Install MongoDB

MongoDB is the database used by nmeta. Install MongoDB as per their instructions (Note: Ubuntu 16.04 specific)

Import the MongoDB public GPG Key:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv␣
→0C49F3730359A14518585931BC711F9BA15703C6
```

Create a list file for MongoDB:

```
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/
→3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list
```

Reload local package database:

```
sudo apt-get update
```

Install MongoDB:

```
sudo apt-get install -y mongodb-org
```

Set MongoDB to autostart:

```
systemctl enable mongod.service
```

Start MongoDB (if required) with:

```
sudo service mongod start
```

Install nmeta

Clone nmeta

```
cd
git clone https://github.com/mattjhayes/nmeta.git
```

Test nmeta

Tests should all pass:

```
cd ~/nmeta/tests/; py.test
```

Run nmeta

Run nmeta in a terminal window:

```
ryu-manager ~/nmeta/nmeta/nmeta.py
```

Run the external API (necessary for WebUI) in a separate terminal window:

```
~/nmeta/nmeta/api_external.py
```

Test WebUI

In a browser on the same machine that nmeta is installed on, navigate to:

http://localhost:8081/webUI/index.html

Configure Switches

Next, we need OpenFlow switches configured to Ryu/nmeta as their controller and app.

Configure OpenFlow

Switches will need to be configured to use Ryu/nmeta as their controller. The configuration details will differ depending on the type of switch.

Here is an example configuration for Open vSwitch to use a controller at IP address 172.16.0.3 on TCP port 6633:

```
sudo ovs-vsctl set-controller br0 tcp:172.16.0.3:6633
```

You will need to work out setting that are appropriate for your topology and switches.

Configure QoS Queues

To run Quality of Service (QoS), switches will need to be configured with QoS queues.

See the documentation for your switch(es) for how to configure QoS queues.

Be aware that using a queue number that is not configured on the switch may result in the switch dropping the packet.

Aliases

Aliases can be used to make it easier to run common commands. To add the aliases, edit the .bash_aliases file in your home directory:

```
cd
sudo vi .bash_aliases
```

Paste in the following:

```
# Test nmeta:
alias nmt='cd ~/nmeta/tests/; py.test'
#
# Run nmeta:
alias nm="ryu-manager ~/nmeta/nmeta/nmeta.py"
#
# Run nmeta external API:
alias nma='~/nmeta/nmeta/api_external.py'
#
# Retrieve Packet-In rate via external API:
alias nma_pi_rate='curl http://localhost:8081/v1/infrastructure/controllers/pi_rate/
→| python -m json.tool'
```

Configure

The nmeta policy configures how nmeta works with data plane traffic. This includes traffic classification rules, what classifiers are used, in what order and what actions are taken.

The policy is designed as a tree with many first level branches and only a shallow depth.

Main Policy

Nmeta ships with a default policy in the YAML file:

```
~/nmeta/nmeta/config/main_policy.yaml
```

Do not edit the default policy as it will be overwritten by nmeta updates.

Create Your Own Policy

Create your own policy by copying the default file to this location:

```
~/nmeta/nmeta/config/user/main_policy.yaml
```

(note the user directory)

If a main_policy.yaml file is present in the user directory it will completely override the default policy. Note that a user-defined main policy file will not be part of the git distribution, as it is excluded in the .gitignore file.

TC Branch - Rules

The traffic classification policy is based off a root key *tc_rules*. This root contains a *ruleset* name (only one ruleset supported at this stage), which in turn contains one or more *rules*. Rules contain *conditions* and these in turn contain classifiers, as per the following diagram:
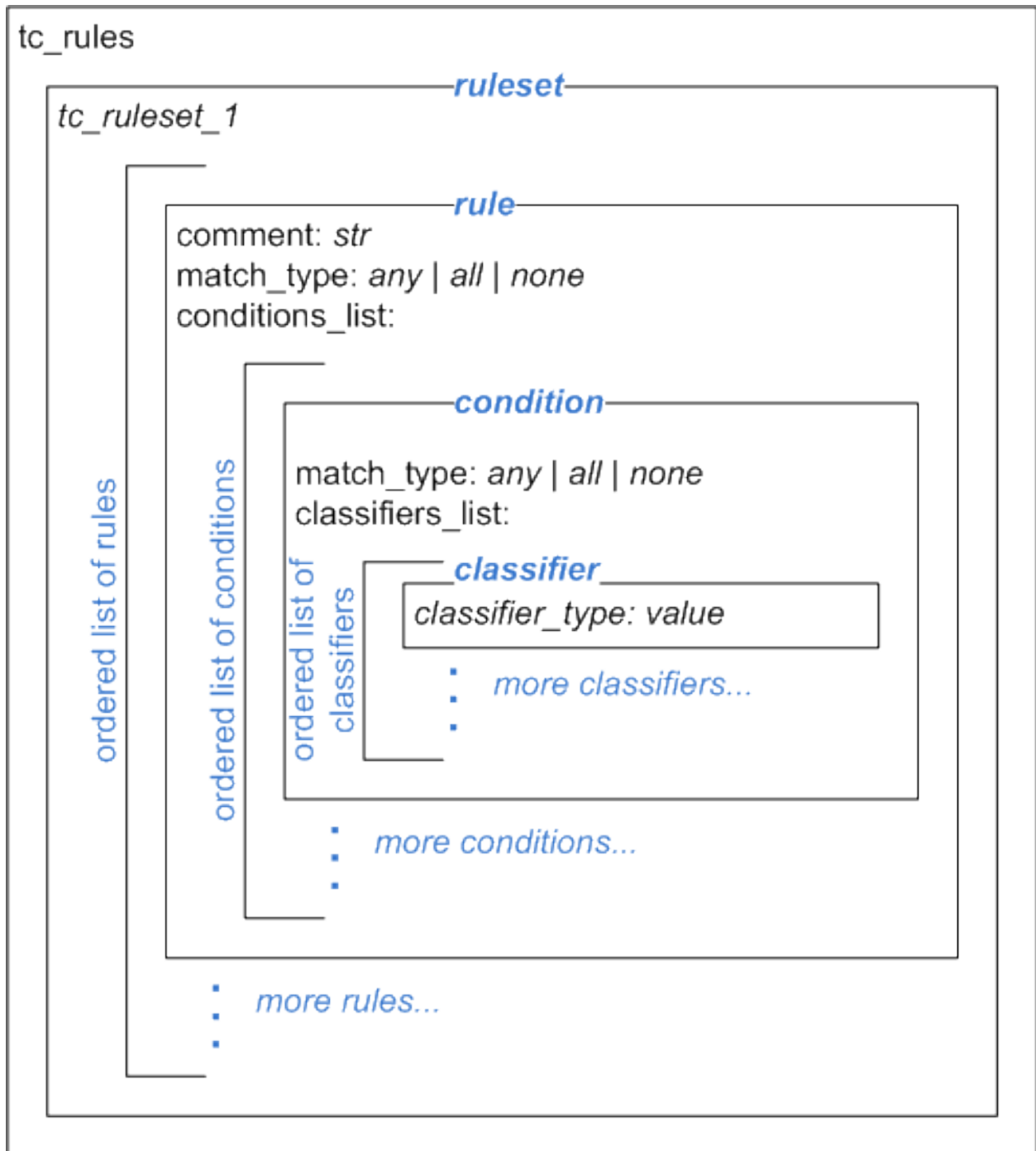
Rules are an ordered list (denoted by preceding dash). Each rule contains:

Comment A *comment* to describe the purpose of the rule (optional). A comment must start with the attribute *comment:* and any single-line string can follow

Match Type A *match_type* is one of:

> **any** Match if any of the conditions in the rule match
>
> **all** Match only if all of the conditions in the rule match
>
> **none** Match only if none of the conditions in the rule match

Conditions List A list that contains one or more *condition* stanzas that each contain a match type and a classifiers_list containing one or more classifiers.

Actions A single *actions* stanza that contains one or more actions

Example simple traffic classification policy with a single rule:

A condition contains:

- **A *match type*, which is one of:**

    any Match if any of the classifiers in the condition match

    all Match only if all of the classifiers in the condition match

    none Match only if none of the classifiers in the condition match.

- A classifiers_list containing one or more *classifiers* (see further below)

An *actions* stanza contains one or more attribute/value pairs

Here is a more complex traffic classification policy:

```
---            Traffic classification rules root
tc_rules:
  # Traffic Classification Rulesets and Rules
  tc_ruleset_1:        Ruleset Name
    #
    # A static rule:
    - comment: OpenFlow Protocol Traffic      Rule signified by preceding list dash
      match_type: any                          Must contain a match_type
      conditions_list:
        - match_type: any      A condition, containing match_type and
          classifiers_list:    classifiers_list
            - tcp_src: 6633
            - tcp_dst: 6633
      actions:                                 Actions Stanza
        set_desc: "OpenFlow Protocol Traffic"
        qos_treatment: high_priority
    #
    # An identity rule:
    - comment: Audit Division SSH traffic
      match_type: all                          Rule
      conditions_list:
        - match_type: any      A condition
          classifiers_list:
            - tcp_src: 22
            - tcp_dst: 22
        - match_type: any                      Second
          classifiers_list:                    condition in list
            - identity_lldp_systemname_re: '.*\.audit\.example\.com'
      actions:
        set_desc: "High Priority Audit SSH Traffic"
        qos_treatment: high_priority           Actions Stanza
    #
    # A custom rule:
    - comment: Constrained Bandwidth Traffic (Statistical)
      match_type: any                          Rule
      conditions_list:
        - match_type: any      A condition
          classifiers_list:
            - custom: 'statistical_qos_bandwidth_1'
      actions:                    Actions Stanza. Custom classifier
        set_desc: classifier_return    returns more than a Boolean
        qos_treatment: classifier_return
```

Conditions invoke classifiers. There are three types of classifier supported:

- Static

- Identity

- Custom (Payload / Statistical / Multi-classifier)

TC Branch - Static Classifiers

Static classifiers match on attributes in packet headers, or on environmental attributes such as port numbers.

Supported attributes are:

location_src

Logical location (as defined by policy) of switch/port

> Example:

```
location_src: external
```

time_of_day

Time of day range (matches if flow start time is in this time range)

> Example:

```
time_of_day: 21:00-07:00
```

Note that range can extend through midnight and times are in 24 hour format

eth_src

Ethernet source MAC address.

> Example:

```
eth_src: 08:00:27:4a:2d:41
```

eth_dst

Ethernet destination MAC address.

> Example:

```
eth_dst: 08:00:27:4a:2d:42
```

eth_type

Ethernet type. Can be in hex (starting with 0x) or decimal.

> Examples:

```
eth_type: 0x0800
```

```
eth_type: 35020
```

ip_src

IP source address. Can be a single address, a network with a mask in CIDR notation, or an IP range with two addresses separated by a hyphen. Both addresses in a range must be the same type, and the second address must be higher than the first.

Examples:

```
ip_src: 192.168.56.12
```

```
ip_src: 192.168.56.0/24
```

```
ip_src: 192.168.56.12-192.168.56.31
```

ip_dst

IP destination address. Can be a single address, a network with a mask in CIDR notation, or an IP range with two addresses separated by a hyphen. Both addresses in a range must be the same type, and the second address must be higher than the first.

Examples:

```
ip_dst: 192.168.57.40
```

```
ip_dst: 192.168.57.0/24
```

```
ip_dst: 192.168.57.36-192.168.78.31
```

tcp_src

TCP source port.

Example:

```
tcp_src: 22
```

tcp_dst

TCP destination port.

Example:

```
tcp_dst: 80
```

udp_src

UDP source port.

Example:

```
udp_src: 123
```

udp_dst

UDP destination port.

> Example:

```
udp_dst: 53
```

TC Branch - Identity Classifiers

All identity classifiers are prefixed with:

```
identity_
```

LLDP systemname may be matched as a regular expression. The match pattern must be contained in single quotes. For example, to match system names of *.audit.example.com, add this policy condition:

```
identity_lldp_systemname_re: '.*\.audit\.example\.com'
```

Supported attributes are:

identity_lldp_systemname

Exact match against a system name discovered via LLDP. Example:

```
identity_lldp_systemname: bob.example.com
```

identity_lldp_systemname_re

Regular expression match against a system name discovered via LLDP. Example:

```
identity_lldp_systemname_re: '.*\.audit\.example\.com'
```

identity_dhcp_hostname

Exact match against a host name discovered via DHCP (option 12). Example:

```
identity_dhcp_hostname: bob
```

identity_dhcp_hostname_re

Regular expression match against a host name discovered via DHCP (option 12). Example:

```
identity_dhcp_hostname_re: 'bob.*'
```

identity_service_dns

Exact match of either IP address in a flow against a DNS domain. Example:

```
identity_service_dns: www.example.com
```

identity_service_dns_re

Regular expression match of either IP address in a flow against a DNS domain. Example:

```
identity_service_dns_re: '.*\.example\.com'
```

TC Branch - Custom Classifiers

Nmeta supports the creation of custom classifiers.

All custom classifiers have the attribute:

```
custom
```

The value determines the custom .py file to load from the nmeta/classifiers directory

For example, the following condition loads a custom classifier file `~/nmeta/nmeta/classifiers/statistical_qos_bandwidth_1.py`:

```
custom: statistical_qos_bandwidth_1
```

TC Branch - Actions

Actions are specific to a rule, and define what nmeta should do when the rule is matched. Multiple actions can be defined on a rule.

Supported attributes are:

drop

Drop the packet

> No flow modification or packet-out will occur. The packet will however appear in metadata and does add load to the controller.
>
> Values can be:
>
> > • at_controller
> >
> > • at_controller_and_switch
>
> Example:

```
drop: at_controller_and_switch
```

> A drop action with 'at_controller_and_switch' value will install a flow entry with no actions (which implicitly drops) onto the switch that sent the matching packet to the controller. Be aware that nmeta will generate a fine-grained match for this drop rule that may not align with what is specified in the policy. It

builds the rule based on the classified packet and will do a match on IPs & TCP or UDP destination port for TCP or UDP or IPs for other IP traffic. It will not apply a rule for non-IP traffic.

qos_treatment

Specify QoS treatment for flow.

Values can be:

- default_priority

- constrained_bw

- high_priority

- low_priority

- classifier_return

Example:

```
qos_treatment: classifier_return
```

set_desc

Set description for the flow. This is a convenience for humans.

Example:

```
set_desc: "This is a flow type description"
```

QoS Treatment Branch

Quality of Service (QoS) treatment parameters are configured in main policy under the qos_treatment root directive. They map qos action values to queue numbers. Example:

```
qos_treatment:
  # Control Quality of Service (QoS) treatment mapping of
  #  names to output queue numbers:
  default_priority: 0
  constrained_bw: 1
  high_priority: 2
  low_priority: 3
```

The QoS queue numbers are arbitrary and are used to map packets and flows to queues that have been configured on the switch (separate to nmeta).

Port Sets Branch

Port Sets are used to abstract a set of switches/ports so that they can be referenced elsewhere in the policy. Port Sets are located under the root key *port_sets*.

Example:

```
port_sets:
    # Port Sets control what data plane ports policies and
    #  features are applied on. Names must be unique.
    port_set_list:
        - name: port_set_location_internal
          port_list:
            - name: VirtualSwitch1-internal
              DPID: 8796748549206
              ports: 1-3,5,66
              vlan_id: 0
            - name: VirtualSwitch2-internal
              DPID: 255
              ports: 3,5
              vlan_id: 0
```

In this example, the port set *port_set_location_internal* refers to specific ports on the switches with DPIDs of 8796748549206 and 255.

Locations Branch

Locations are a policy-defined aspect of an identity that are based on the source or destination DPID/port, which is looked up against a list that links location names to port sets.

Locations are located under the root key *locations*.

A default location must be defined.

Example:

```
locations:
    # Locations are logical groupings of ports. Takes first match.
    locations_list:
        - name: internal
          port_set_list:
            - port_set: port_set_location_internal
        - name: external
          port_set_list:
            - port_set: port_set_location_external
    default_match: unknown
```

System Config

A YAML file holds the system configuration. You wouldn't normally need to change this file from the defaults. It allows you to change values like timers, database sizing and logging levels.

It's location is:

```
~/nmeta/nmeta/config/config.yaml
```

These default configuration parameters can be overwritten by creating a file:

```
~/nmeta/nmeta/config/user/config.yaml
```

Add the parameters to the file that you want to override. For example, to override the default console logging level for the tc_policy module, add the following line to the user config file:

```
tc_policy_logging_level_c: INFO
```

Note that the user-defined config file will not be part of the git distribution, as it is excluded in the .gitignore file.

Build a Lab

To run nmeta, you're going to need an OpenFlow network to provide the data plane connectivity.

There are many different options for building a lab network. The choice is likely to come down to what resources you have and the use cases you want to test.

Virtual labs are easy to set up and don't require specialised hardware, but aren't useful for testing devices in the real world.

Physical labs are harder to construct and require hardware, but can be used to connect real-world devices.

OpenFlow SDN disaggregates the data and control planes; this means the lab environments can be used with different OpenFlow controllers and apps, should you wish.

Physical Labs

OpenWRT with Open vSwitch

This lab is based on a TP-Link TL-WR1043ND Hardware Version 2.1 home router that is re-flashed to run OpenWRT with Open vSwitch running OpenFlow (yes, that's three different pieces of software that start with the word 'Open'...)

Be warned that reflashing a router is likely to void it's warrantee, and may result in the router becoming 'bricked', whereby it is unrecoverable. Continue **at your own risk**...

You'll also need a physical Linux PC with two NICs that has been built with nmeta as per the install instructions.

The configuration of the lab is shown below:
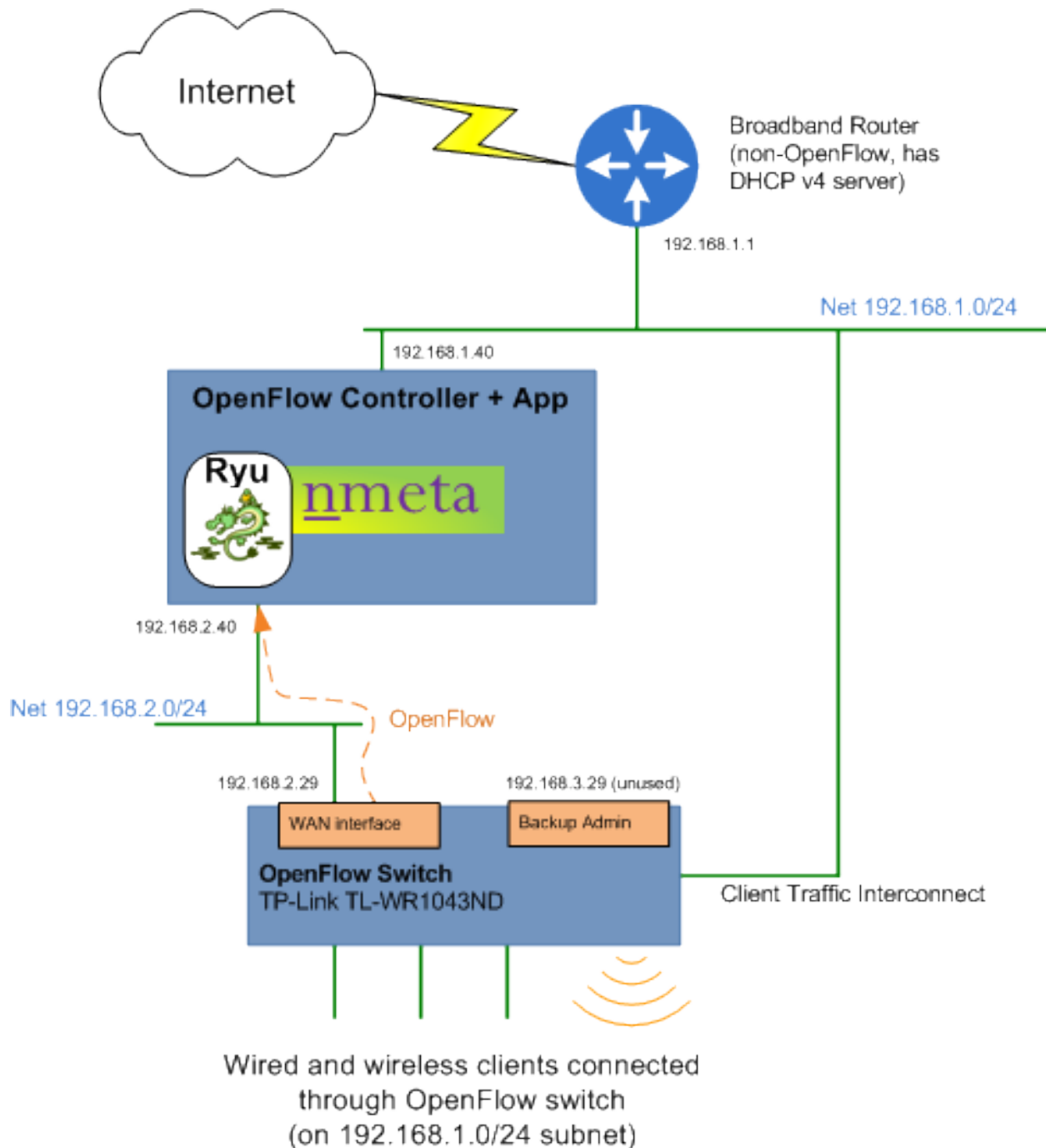
These instructions haven't been tested end-to-end. Please raise an issue if there are changes required.

Convert Router to OpenWRT

Start by converting the TP-Link TL-WR1043ND to running OpenWRT as per the instructions from the OpenWRT website at:

https://wiki.openwrt.org/toh/tp-link/tl-wr1043nd

When router is successfully running OpenWRT, proceed to the next step:

Configure the Router

Apply a basic configuration to the router to allow remote access.

Connect a device with SSH capability to a LAN port on the TP-Link, set a static IP address of 192.168.1.2 mask 255.255.255.0 (or use DHCP) and SSH to 192.168.1.1.

Set root password to something secure, and not used elsewhere.

Compile OpenWRT with Open vSwitch Image

Note: If you don't want to compile your own image then consider using an image from https://github.com/mattjhayes/ TP-Link-TL-1043ND-OpenvSwitch and jump ahead to http://nmeta.readthedocs.io/en/develop/userguide/build_a_ lab.html#upgrade

Compilation Host

To compile the router firmware, use an Ubuntu 16.04.2 server or desktop (can be virtual) with at least 30GB of disk space.

Clone OpenWRT

On the compilation host, clone OpenWRT (note: GitHub, not direct from OpenWRT site):

```
git clone https://github.com/openwrt/openwrt.git
```

Install Dependancies

```
sudo apt-get update
sudo apt-get install git-core build-essential libssl-dev libncurses5-dev unzip gawk␣
↪zlib1g-dev
sudo apt-get install subversion mercurial
sudo apt-get install gcc-multilib flex gettext
```
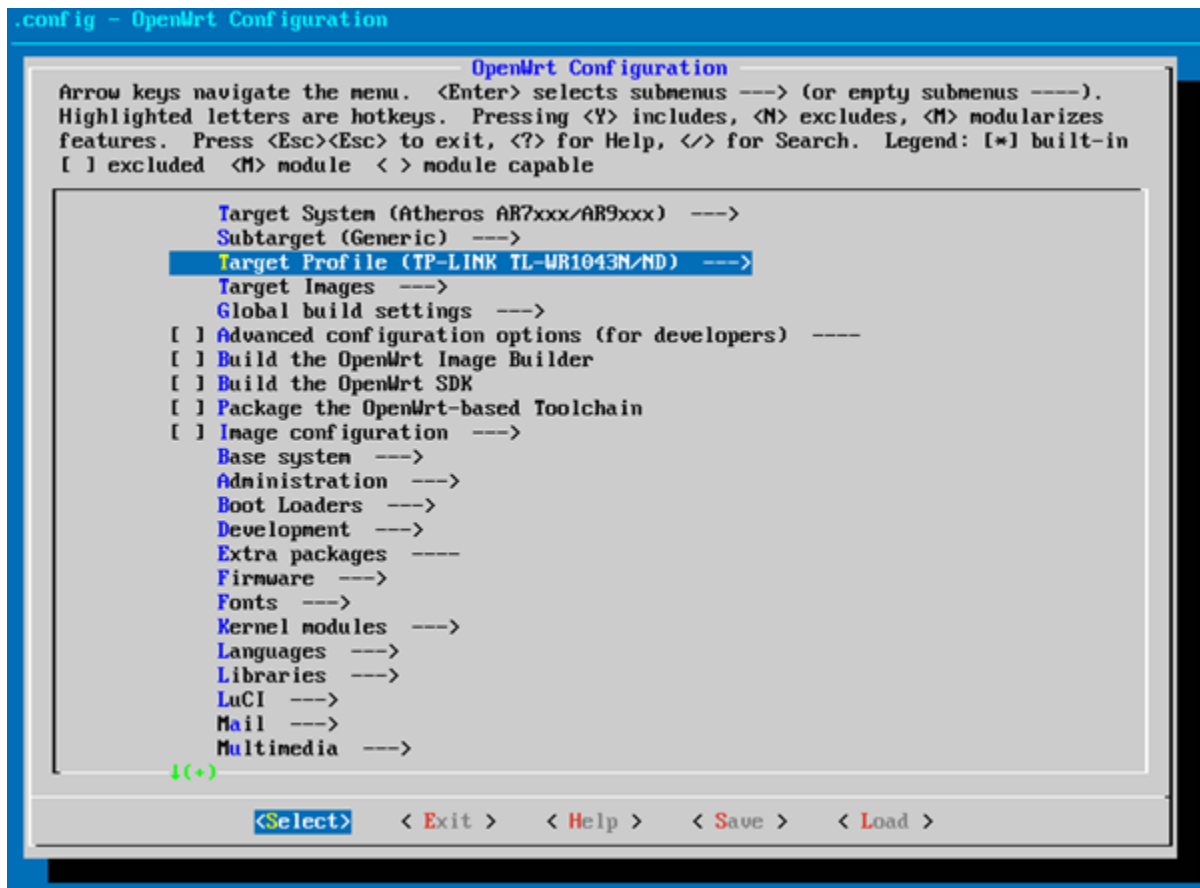
Update Feeds

```
cd openwrt
./scripts/feeds update -a
./scripts/feeds install -a
```

Make MenuConfig

```
make menuconfig
```

Change Target Profile to suit hardware (select *TP-LINK TL-WR1043N/ND* for TP-Link TL-WR1043ND Hardware Version 2.1):

Then select Kernel Modules -> Network Support -> kmod-tun:
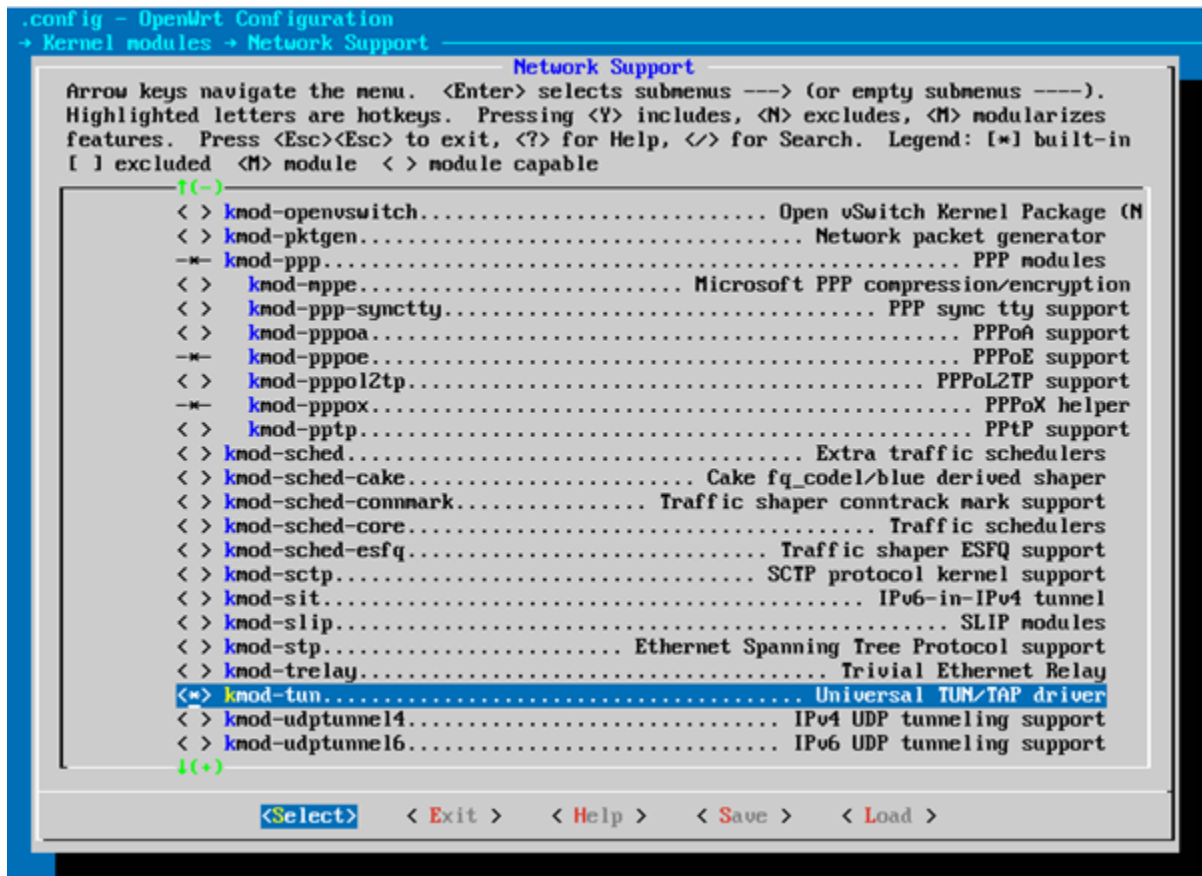
Exit out back to main screen, then select *Network -> Open vSwitch* and select:
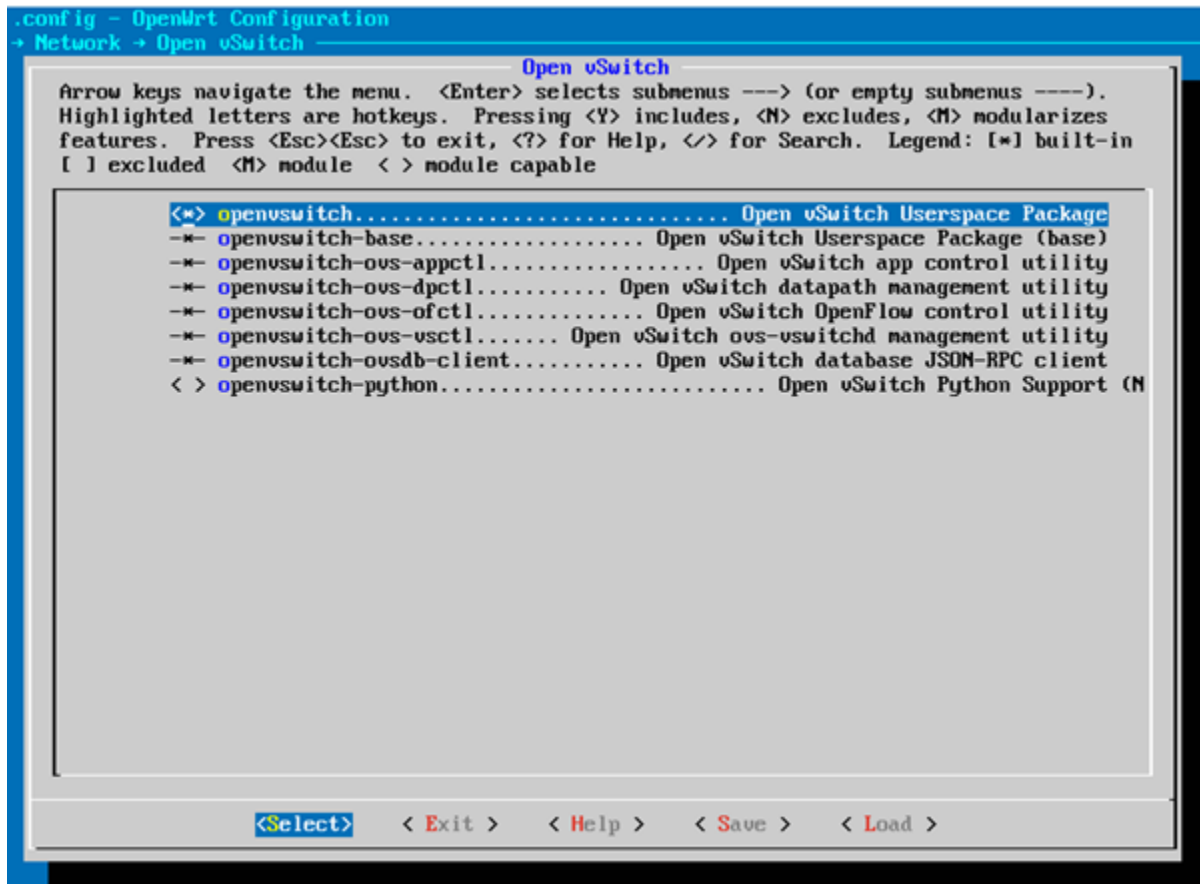
Save on exit:

This one takes a while:

```
make kernel_menuconfig
```

When finished brings up another menu. Navigate to *Networking support -> Networking options* and select *Hierarchical Token Bucket (HTB)*:

```
.config - Linux/mips 4.4.14 Kernel Configuration
 → Networking support → Networking options
                            Networking options
    Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
    Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
    features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
    [ ] excluded  <M> module  < > module capable
            ↑(-)
             [ ]    VLAN filtering
             <*> Distributed Switch Architecture
             <*> 802.1Q/802.1ad VLAN Support
             [ ]    GVRP (GARP VLAN Registration Protocol) support
             [ ]    MVRP (Multiple VLAN Registration Protocol) support
             < > DECnet Support
             < > ANSI/IEEE 802.2 LLC type 2 Support
             < > The IPX protocol
             < > Appletalk protocol support
             < > CCITT X.25 Packet Layer
             < > LAPB Data Link Driver
             < > Phonet protocols family
             < > IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks support  ----
                 *** Queueing/Scheduling ***
             < > Class Based Queueing (CBQ)
             <*> Hierarchical Token Bucket (HTB)
             < > Hierarchical Fair Service Curve (HFSC)
             < > Multi Band Priority Queueing (PRIO)
             < > Hardware Multiqueue-aware Multi Band Queueing (MULTIQ)
             < > Random Early Detection (RED)
             < > Stochastic Fair Blue (SFB)
             < > Stochastic Fairness Queueing (SFQ)
             < > Enhanced Stochastic Fairness Queueing (ESFQ)
            ↓(+)

                <Select>     < Exit >    < Help >    < Save >    < Load >
```
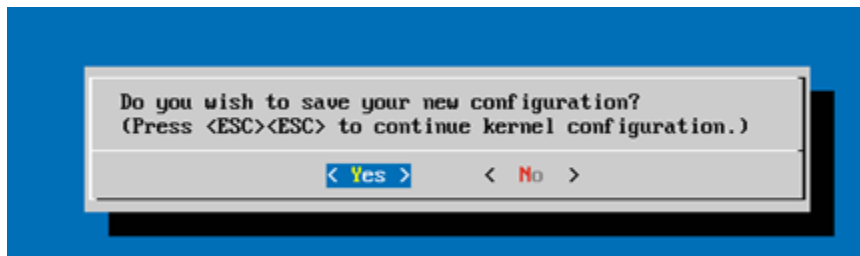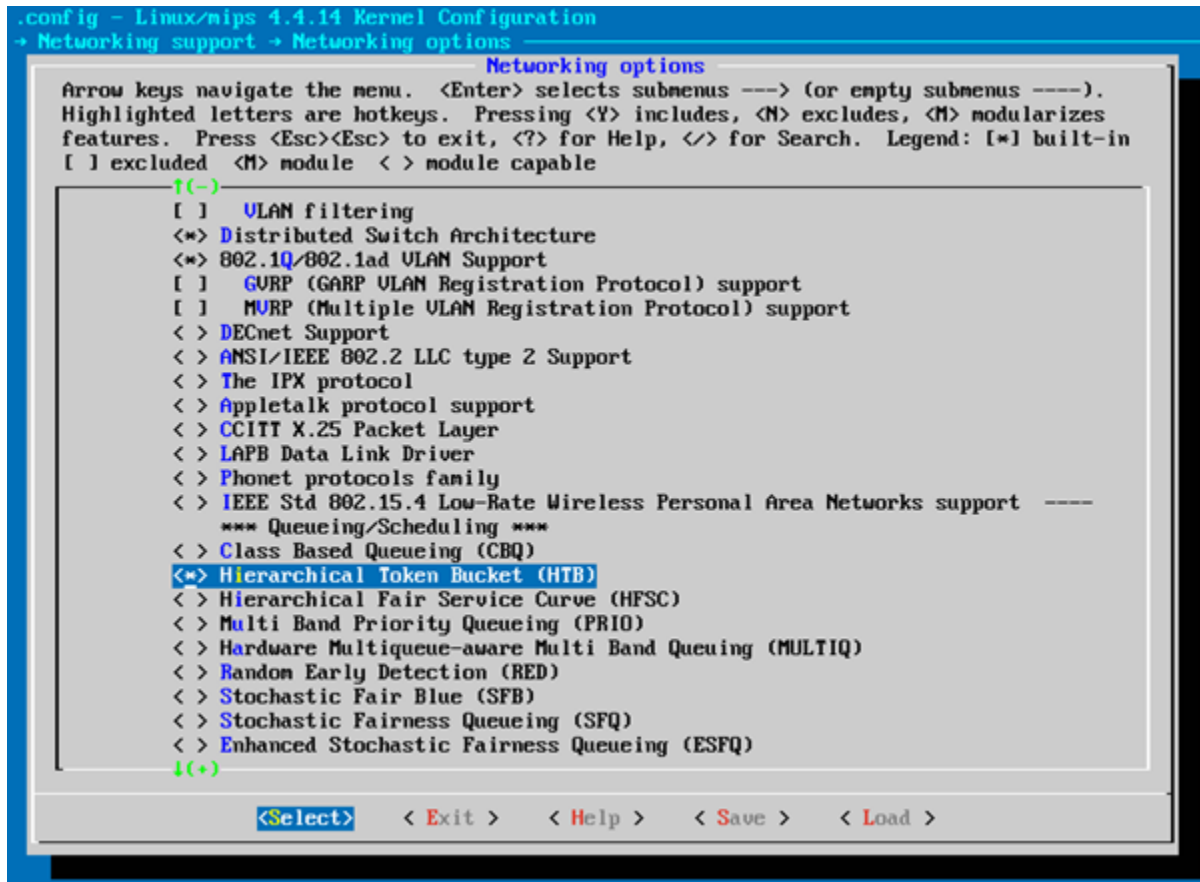
Run Make

This may take a couple of hours...

```
make
```

Patch for Wi-Fi Authentication

Standard OpenWRT build with Open vSwitch cannot run authentication on Wi-Fi, see: https://forum.openwrt.org/viewtopic.php?id=59129

We apply a patch to fix this:

```
cd ~/openwrt/package/network/services/hostapd/
vi 710-hostapd-Initial-OVS-support.patch
```

Paste in contents of patch (starting from the —) from https://github.com/helmut-jacob/hostapd/commit/c89daaeca4ee90c8bc158e37acb1b679c823d7ab.patch Save and exit.

Patch with Quilt. Install quilt:

```
sudo apt install quilt
```

In home dir, need to run this once:

```
cat > ~/.quiltrc <<EOF
QUILT_DIFF_ARGS="--no-timestamps --no-index -p ab --color=auto"
QUILT_REFRESH_ARGS="--no-timestamps --no-index -p ab"
QUILT_SERIES_ARGS="--color=auto"
QUILT_PATCH_OPTS="--unified"
QUILT_DIFF_OPTS="-p"
EDITOR="nano"
EOF
```

Run this from ~/openwrt/

```
make package/network/services/hostapd/{clean,prepare} V=s QUILT=1
```

cd to created directory:

```
cd ~/openwrt/build_dir/target-mips_34kc_musl-1.1.16/hostapd-wpad-mini/hostapd-2016-06-
↪15/
```

Apply existing patches:

```
quilt push -a
```

Now at patch 710-hostapd-Initial-OVS-support.patch. Run this:

```
quilt edit src/main.c
```

Run this:

```
quilt refresh
```

Change dir to the build root and run

```
cd ../../../../
make package/network/services/hostapd/update V=s
```

Then run:

```
make package/network/services/hostapd/{clean,compile} package/index V=s
```

Then run:

```
make
```

Copy Image

Navigate to the directory where the output files are:

```
cd bin/ar71xx
```

There should be multiple files in the directory, including this file:

```
openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin
```

Use SCP to copy the appropriate file to the router:

```
scp ./openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin USERNAME@192.168.
↪1.1:tmp/
```

### Upgrade

Note: consider backing up config etc first...

Once image file is confirmed as being in the /tmp directory on the TPLink, and you're happy you've backed up your configurations, run the sysupgrade:

```
sysupgrade -v /tmp/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin
```

Configure OpenWRT

OpenWRT needs to be configured to work with Open vSwitch. The configuration has been tested, but needs to be changed to meet your requirements.

The following diagram shows how OpenWRT with Open vSwitch is configured on the TP-Link hardware:
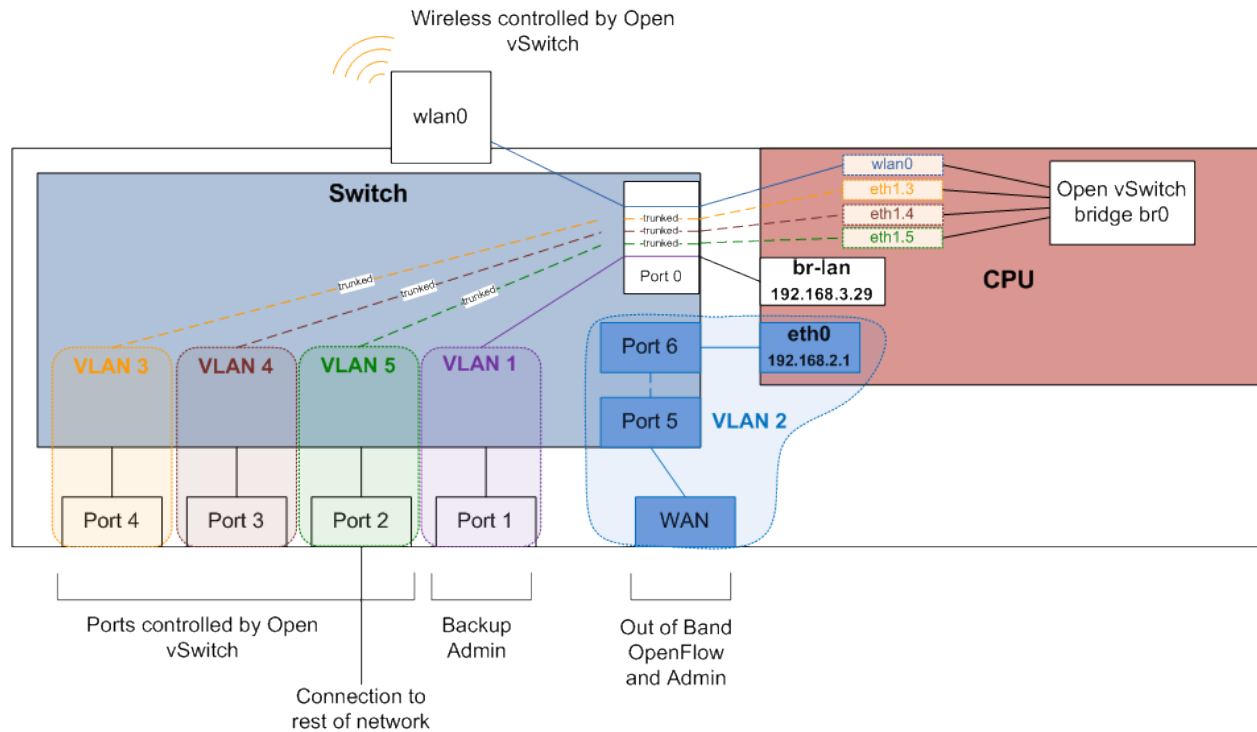
Dropbear

Configure Dropbear (SSH server) to listen on the WAN interface, in addition to the LAN interface. This gives an additional way to access the device to administer it, lowering the risk of bricking it.

Note: not a great idea doing this if Internet-facing for security reasons, so remember to remove WAN config if you ever convert device back to an Internet router.

Backup dropbear config:

```
cp /etc/config/dropbear /etc/config/dropbear.original
```

Add these lines to /etc/config/dropbear for WAN, full file is:

```
config dropbear
        option PasswordAuth 'on'
        option Port '22'
        option Interface 'lan'

config dropbear
        option PasswordAuth 'on'
        option Port '22'
        option Interface 'wan'
```

Firewall

Firewall (/etc/config/firewall) should be default permissive policy:

```
config defaults
        option syn_flood        1
        option input            ACCEPT
        option output           ACCEPT
        option forward          ACCEPT
```

Network

Backup network config:

```
cp /etc/config/network /etc/config/network.original
```

This is the new complete */etc/config/network* file:

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'lan'
        option ifname 'eth1'
        option force_link '1'
        option type 'bridge'
        option proto 'static'
        option ipaddr '192.168.3.29'
        option netmask '255.255.255.0'

config interface 'wan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.2.29'
        option netmask '255.255.255.0'
        option defaultroute '1'
        option gateway '192.168.2.40'
        option dns '8.8.8.8'
```

```
config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '0 4'

config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '5 6'

config switch_vlan
        option device 'switch0'
        option vlan '3'
        option ports '0t 1'

config switch_vlan
        option device 'switch0'
        option vlan '4'
        option ports '0t 2'

config switch_vlan
        option device 'switch0'
        option vlan '5'
        option ports '0t 3'

config interface
        option ifname 'eth1.3'
        option proto 'static'
        option ipv6 '0'

config interface
        option ifname 'eth1.4'
        option proto 'static'
        option ipv6 '0'

config interface
        option ifname 'eth1.5'
        option proto 'static'
        option ipv6 '0'

config interface 'wan6'
        option proto 'dhcpv6'
        option ifname '@wan'
        option reqprefix 'no'

config interface
        option ifname 'br0'
        option proto 'static'

config interface
        option ifname 'wlan0'
        option proto 'static'
```

Wireless

Backup wireless config:

```
cp /etc/config/wireless /etc/config/wireless.original
```

Take note of the items in CAPITALS that need you to fill in appropriate values. This is the new complete */etc/config/wireless* file:

```
config wifi-device 'radio0'
        option type 'mac80211'
        option channel '11'
        option hwmode '11g'
        option path 'platform/qca955x_wmac'
        option htmode 'HT20'
        option log_level '1'

config wifi-iface
        option device 'radio0'
        option network 'wlan0'
        option mode 'ap'
        option ssid 'YOUR_SSID_HERE'
        option encryption 'psk2'
        option key 'YOUR_KEY_HERE'
```

Configure Open vSwitch

Now it's time to configure Open vSwitch by setting up bridge *br0*, adding ports to it, then setting it to talk OpenFlow to the Controller:

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth1.3
ovs-vsctl add-port br0 eth1.4
ovs-vsctl add-port br0 eth1.5
ovs-vsctl add-port br0 wlan0
ovs-vsctl set-controller br0 tcp:192.168.2.40:6633
```

Configure Aliases

Aliases are useful for frequently run commands. Here are some suggested aliases.

Edit file */etc/profile* and add these lines:

```
# OpenWRT Network Commands:
alias nwr='/etc/init.d/network restart'

# Open vSwitch Commands:
alias ovshow='ovs-vsctl show'
alias ovmacs='ovs-appctl fdb/show br0'
alias ovrestart='/etc/init.d/openvswitch restart'

# Open vSwitch OpenFlow Commands:
alias ofshow='ovs-ofctl show br0'
alias offlows='ovs-ofctl dump-flows br0'
alias ofports='ovs-ofctl dump-ports br0'
```

Log out and back in again to enable new aliases.

Cabling

Wire the environment together as per earlier diagram, and ensure the Linux PC has it's network interfaces configured correctly.

Checks

Using our aliases, here are checks to run:

```
# ovshow
<snip>
  Bridge "br0"
      Controller "tcp:192.168.2.40:6633"
          is_connected: true
      Port "br0"
          Interface "br0"
              type: internal
      Port "wlan0"
          Interface "wlan0"
      Port "eth1.3"
          Interface "eth1.3"
      Port "eth1.4"
          Interface "eth1.4"
      Port "eth1.5"
          Interface "eth1.5"
```

Note the *is_connected: true*. This means OpenFlow has been established to the controller.

Links

Instructions were based on these tutorials:

Building and Configuring Open vSwitch on OpenWrt for Cloud Networking byPravin R. Turning TP-LINK WR1043NDv2.1 router into OpenFlow-enabled switch by Lucas Burson

Virtual Labs

Mininet with Vagrant

UNDER CONSTRUCTION

In this lab we use Vagrant to automate the start up and build of multiple VirtualBox Ubuntu guests.

These instructions assume you're running Windows, but should be easily adapted to other operating systems as most of the work happens within the virtual environment.

Install VirtualBox

Download and install VirtualBox from https://www.virtualbox.org/wiki/Downloads

Install Vagrant

Download and install Vagrant from https://www.vagrantup.com/

Download a box

We will use the bento box of Ubuntu 16.04 in this lab. Download this box on your host machine with:

```
vagrant box add bento/ubuntu-16.04
```

Choose *virtualbox* option from menu

Clone Vagrant Repo

Clone the Vagrant repo from https://github.com/mattjhayes/Vagrant onto your host machine.

Start the Guest

In a command prompt, from base of cloned repo, navigate to the *SDN_LabsMininet_Ryu_nmeta* directory:

```
cd SDN_Labs\Mininet_Ryu_nmeta
```

Start the guest by running this on the host machine command prompt:

```
vagrant up
```

When the guest is up, connect to it with SSH on localhost:2222

username/password are both *vagrant*

run nmeta (from alias):

```
nm
```

Start a second SSH session and run the nmeta api:

```
nma
```

In a third SSH session run Mininet:

```
mnt
```

TBD - UNDER CONSTRUCTION

VirtualBox with Vagrant

UNDER CONSTRUCTION

In this lab we use Vagrant to automate the start up and build of multiple VirtualBox Ubuntu guests.

These instructions assume you're running Windows, but should be easily adapted to other operating systems as most of the work happens within the virtual environment.

Install VirtualBox

Download and install VirtualBox from https://www.virtualbox.org/wiki/Downloads

Install Vagrant

Download and install Vagrant from https://www.vagrantup.com/

Download a box

We will use the bento box of Ubuntu 16.04 in this lab. Download this box on your host machine with:

```
vagrant box add bento/ubuntu-16.04
```

Choose *virtualbox* option from menu

Clone Vagrant Repo

Clone the Vagrant repo from https://github.com/mattjhayes/Vagrant onto your host machine.

Start the Guest

In a command prompt, from base of cloned repo, navigate to the *SDN_LabsRyu_nmeta_SystemTestLab* directory:

```
cd SDN_Labs\Ryu_nmeta_SystemTestLab
```

Start the guest by running this on the host machine command prompt:

```
vagrant up
```

When the guests are up, connect to the controller with SSH on localhost:2203 (first guests is port 2222 then ports 2200 and upwards for other guests)

username/password are both *vagrant*

run nmeta (from alias):

```
nm
```

Start a second SSH session and run the nmeta api:

```
nma
```

TBD - UNDER CONSTRUCTION

Web UI

The nmeta web UI provides a graphical interface into network metadata. It is currently under construction, so functionality is limited and results will vary...

To use the web UI, start nmeta (alias nm), start the nmeta external API (alias nma) and then point a local (doesn't have to be local) browser at:
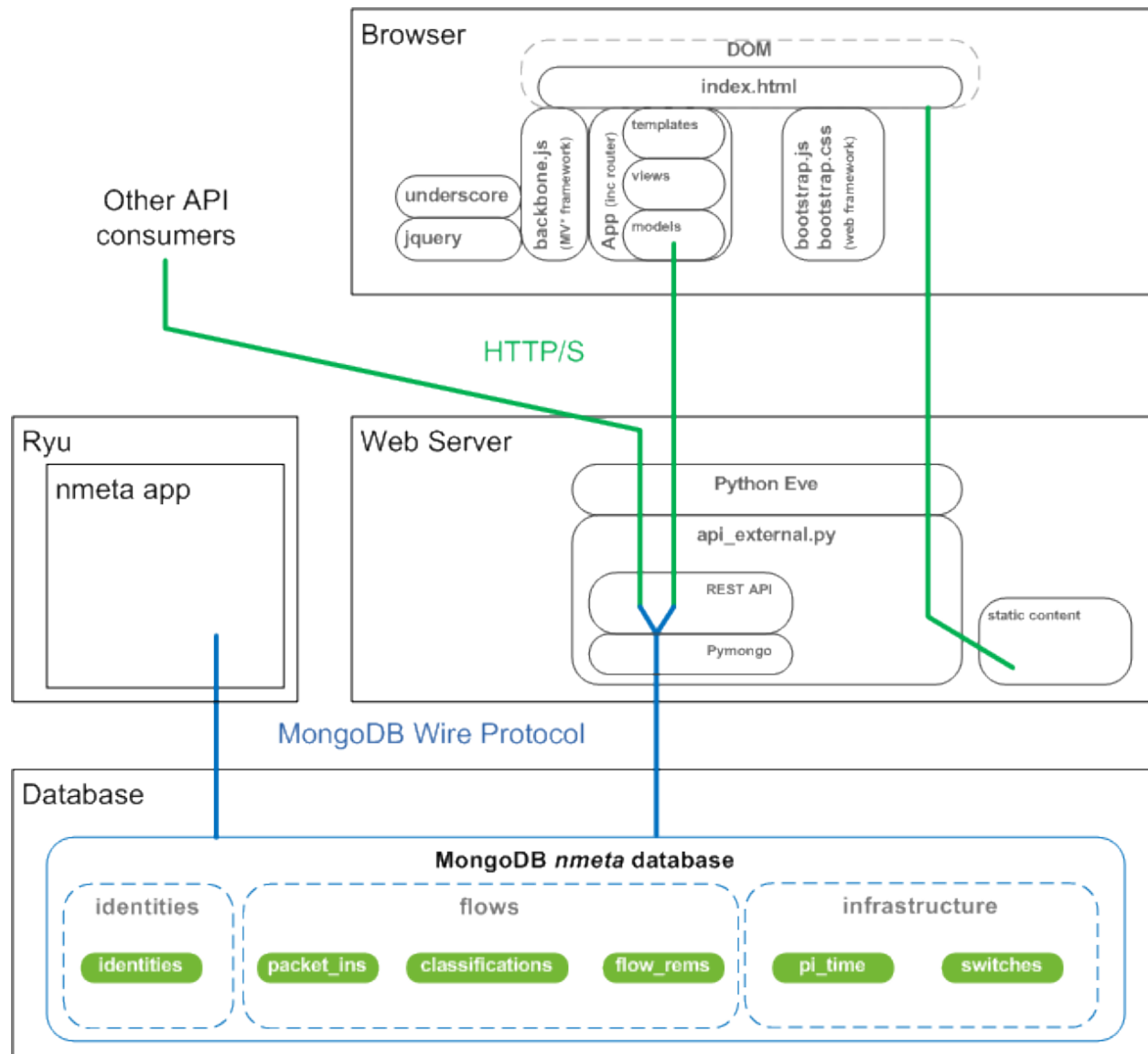
http://localhost:8081/webUI/index.html

The home page looks like this:
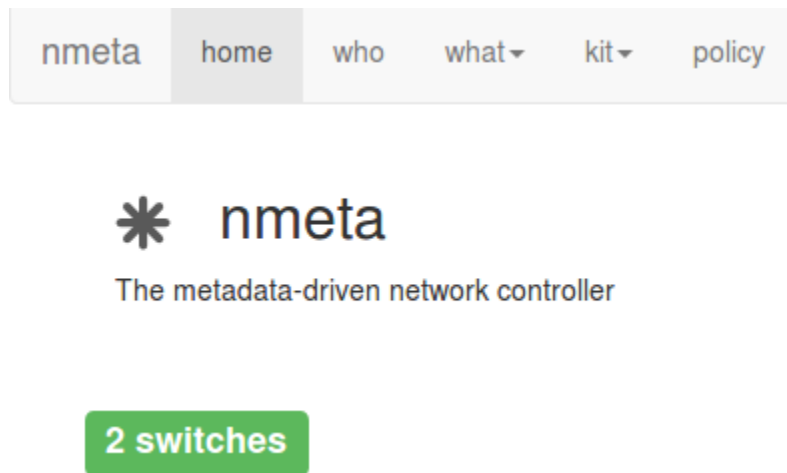
The number of connected switches updates dynamically.

The architecture of the WebUI and REST interface are shown in the diagram below:

The Web Server, Ryu/nmeta and the MongoDB database all run independently. Backbone.js is the JavaScript framework used to power the UI in the browser. Bootstrap is the web framework used to style the presentation of the UI.

APIs

Nmeta uses Python Eve to expose various RESTful API types:

- *Flow APIs*
- *Identity APIs*
- *Infrastructure APIs*
- *Internal APIs*

Flow APIs

Flow Mods API

The Flow Mods API is a read-only summary of all flow modifications made to switches by the controller.

It is a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/flow_mods_api.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flow_mods/ | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [

        {
            "_created": "00:00:00.000000",
            "_etag": "21a20685ccf9080fbd31de81eb2802146907bf13",
```

```
            "_id": "59d807e101186126d01fc216",
            "_updated": "00:00:00.000000",
            "dpid": 1,
            "flow_hash": "c907986d4796fb669acb37efba3afc8e",
            "forward_cookie": 1,
            "forward_match": {
                "eth_type": 2048,
                "ip_proto": 6,
                "ipv4_dst": "10.1.0.1",
                "ipv4_src": "10.1.0.2",
                "tcp_dst": 36296,
                "tcp_src": 80
            },
            "match_type": "dual",
            "reverse_cookie": 2,
            "reverse_match": {
                "eth_type": 2048,
                "ip_proto": 6,
                "ipv4_dst": "10.1.0.2",
                "ipv4_src": "10.1.0.1",
                "tcp_dst": 80,
                "tcp_src": 36296
            },
            "standdown": 0,
            "suppress_type": "suppress",
            "timestamp": "11:46:57.940000"
        },
```

Flows API

The Flows API is a read-only summary of all flows recorded by the controller.

It is a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/flows_api.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows/ | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
            "_etag": "6fbc72e6d279932c763db5852312ccd4b4f6d4cc",
            "_id": "59d81f3a0118612dd314c8b0",
            "_updated": "00:00:00.000000",
            "client_ip": "10.1.0.1",
            "dpid": 2,
            "flow_hash": "3c1a773547e36469500f64ad0b34efb2",
            "forward_cookie": 1,
            "forward_match": {
                "eth_type": 2048,
```

```
            "ip_proto": 6,
            "ipv4_dst": "10.1.0.2",
            "ipv4_src": "10.1.0.1",
            "tcp_dst": 80,
            "tcp_src": 36299
        },
        "match_type": "dual",
        "reverse_cookie": 2,
        "reverse_match": {
            "eth_type": 2048,
            "ip_proto": 6,
            "ipv4_dst": "10.1.0.1",
            "ipv4_src": "10.1.0.2",
            "tcp_dst": 36299,
            "tcp_src": 80
        },
        "standdown": 0,
        "suppress_type": "suppress",
        "timestamp": "13:26:34.546000"
    }
```

Flows UI API

The Flows UI API is a read-only summary of all flows recorded by the controller, tailored for use by the WebUI. It features the following: - Flow direction normalised to direction of first packet in flow - Src and Dst are IP or Layer 2 to optimise screen space - Extra data included for hover-over tips - Enriched with classification and action(s) - Enriched with data xfer (only applies to flows that have had idle timeout)

It is not a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/flows_ui.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows/ui/ | python -m json.tool
```

Flows Removed API

The Flows Removed API is a read-only summary of all removed flows recorded by the controller (switches send flow removal messages to the controller). It does not deduplicate for same flow being removed from multiple switches.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/flows_removed_api.py
```

Flows Removed API

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows_removed/ | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
            "_etag": "4c6fba64b571e392f578aa6804b5ad45149a1b5c",
            "_id": "59b3213f01186111d817494c",
            "_updated": "00:00:00.000000",
            "byte_count": 468,
            "cookie": 5,
            "dpid": 1,
            "duration_sec": 31,
            "eth_A": "",
            "eth_B": "",
            "eth_type": 2048,
            "flow_hash": "fada031e16b76ef92e68aa516123c500",
            "hard_timeout": 0,
            "idle_timeout": 30,
            "ip_A": "10.1.0.1",
            "ip_B": "10.1.0.2",
            "ip_proto": 6,
            "packet_count": 7,
            "priority": 1,
            "reason": 0,
            "removal_time": "11:01:19.121000",
            "table_id": 0,
            "tp_A": 45593,
            "tp_B": 80
        },
```

Flows Removed Stats Count

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows_removed/stats/count | python -m json.tool
```

Example response:

```
{
    "flows_removed": 4
}
```

Flows Removed Stats Bytes Sent

Aggregates and sums byte_count by source IP address. Deduplicates for same flow hash removed from multiple switches and reverse sorts by bytes

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows_removed/stats/bytes_sent | python -m json.tool
```

Example response:

```
{
    "_items": [
        {
            "_id": "10.1.0.2",
            "identity": "10.1.0.2",
            "total_bytes_sent": 3532
        },
        {
            "_id": "10.1.0.1",
            "identity": "pc1",
            "total_bytes_sent": 1404
        }
    ]
}
```

Flows Removed Stats Bytes Received

Aggregates and sums byte_count by destination IP address. Deduplicates for same flow hash removed from multiple switches and reverse sorts by bytes

Example manual invocation of the API:

```
curl http://localhost:8081/v1/flows_removed/stats/bytes_received | python -m json.tool
```

Example response:

```
{
    "_items": [
        {
            "_id": "10.1.0.1",
            "identity": "pc1",
            "total_bytes_received": 3532
        },
        {
            "_id": "10.1.0.2",
            "identity": "10.1.0.2",
            "total_bytes_received": 1404
        }
    ]
}
```

Classifications

The classifications API returns the results of traffic classifications on flows.

Example manual invocation of the API:

```
curl http://localhost:8081/v1/classifications | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
```

```
            "_etag": "2edf91b82d854695895ee44cffbcd5886209d12b",
            "_id": "59f4e7f2011861131aea939a",
            "_updated": "00:00:00.000000",
            "actions": {
                "qos_treatment": "constrained_bw",
                "set_desc": "Constrained Bandwidth Traffic"
            },
            "classification_tag": "Constrained Bandwidth Traffic",
            "classification_time": "09:26:26.131000",
            "classified": true,
            "flow_hash": "7af8ea9080506199633414caba6259e6"
        },
```

Identity APIs

Identities API

The Identities API is a read-only summary of all identity records harvested by the controller.

It is a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/identities_api.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/identities/ | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
            "_etag": "79b7626eba366805e4723ce81751c100b447d04c",
            "_id": "59b3206801186111d817487b",
            "_updated": "00:00:00.000000",
            "dpid": 2,
            "harvest_time": "10:57:43.997000",
            "harvest_type": "ARP",
            "host_desc": "",
            "host_name": "",
            "host_os": "",
            "host_type": "",
            "id_hash": "aafeaa6798c9ef3761f7afe51dd3cf7d",
            "in_port": 2,
            "ip_address": "10.1.0.1",
            "mac_address": "08:00:27:2a:d6:dd",
            "service_alias": "",
            "service_name": "",
            "user_id": "",
            "valid_from": "10:57:43.997000",
            "valid_to": "14:57:43.997000"
        },
```

Identities UI API

The Identities API is a read-only summary of all identity records harvested by the controller, tailored for use by the WebUI. It features the following: - Reverse sort by harvest time - Deduplicate by id_hash, only returning most recent per id_hash - Includes possibly stale records - Checks DNS identities to see if they are from a CNAME, and if so includes

> IP address from the A record

- Optional filtering out of DNS identities by setting '?filter_dns=1' on URI

It is not a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/identities_ui.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/identities/ui/ | python -m json.tool
```

Example response (showing only one of multiple records):

```
{
    "_items": [
        {
            "_id": "59b31fc301186111d81747ae",
            "dpid": 1,
            "harvest_time": "10:54:59.131000",
            "harvest_type": "LLDP",
            "host_desc": "Ubuntu 16.04.2 LTS Linux 4.4.0-93-generic #116-Ubuntu SMP
→Fri Aug 11 21:17:51 UTC 2017 x86_64",
            "host_name": "sw2.example.com",
            "host_os": "",
            "host_type": "",
            "id_hash": "ab044209ef247d208ca1e88c5727ba0c",
            "in_port": 2,
            "ip_address": "",
            "location_logical": "internal",
            "location_physical": "",
            "mac_address": "08:00:27:ea:23:84",
            "service_alias": "",
            "service_name": "",
            "user_id": "",
            "valid_from": "10:54:59.131000",
            "valid_to": "10:56:59.131000"
        },
```

Infrastructure APIs

APIs expose nmeta performance and state data. They are used by the nmeta WebUI and can be used for other applications.

Be aware that some non-native Python Eve APIs have limited feature support (i.e. may not support filtering)

Controller Summary API

The Controller Summary API is a read-only summary of the current controller performance metrics.

It is not a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/controller_summary.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/infrastructure/controllers/summary/ | python -m json.
↪tool
```

PI Rate API

The PI Rate API is a read-only metric for the rate at which the controller is receiving packet-in (PI) messages.

It is not a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/pi_rate.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/infrastructure/controllers/pi_rate/ | python -m json.
↪tool
```

Example response:

```
{
    "pi_rate": 0.2,
    "timestamp": "19:21:35"
}
```

PI Time API

The PI Time API is a read-only set of metrics for the timeliness of the controller in processing packet-in (PI) messages. It is measured over the length of time defined by PACKET_TIME_PERIOD, as defined in api_external.py, and returned in the API as the key pi_time_period.

It is not a native Python Eve API.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/pi_time.py
```

Example manual invocation of the API:

```
curl http://localhost:8081/v1/infrastructure/controllers/pi_time/ | python -m json.
↪tool
```

Example response:

```
{
"pi_time_avg": 0.05947005748748779,
"pi_time_max": 0.06364011764526367,
"pi_time_min": 0.055299997329711914,
"pi_time_period": 10,
"pi_time_records": 2,
"ryu_time_avg": 0.0007699728012084961,
"ryu_time_max": 0.0008089542388916016,
"ryu_time_min": 0.0007309913635253906,
"ryu_time_period": 10,
"ryu_time_records": 2,
"timestamp": "19:50:40"
}
```

Switches API

The Switches API provides information on switches connected to the controller.

The API definition file is at:

```
~/nmeta/nmeta/api_definitions/switches_api.py
```

Switch Details

The Switch Details API is a read-only summary of all switches currently connected to controller.

Example manual invocation of the API:

```
curl http://localhost:8081/v1/infrastructure/switches/ | python -m json.tool
```

Example response:

```
{
    "_items": [
        {
            "_created": "00:00:00.000000",
            "_etag": "e9cf4f29afa425bc0486cda334c56017d3d6e2ca",
            "_id": "59854e3ee14ebffa9f4f4e7b",
            "_updated": "00:00:00.000000",
            "dp_desc": "None",
            "dpid": 1,
            "hw_desc": "Open vSwitch",
            "ip_address": "172.16.0.5",
            "mfr_desc": "Nicira, Inc.",
            "port": 46074,
            "serial_num": "None",
            "sw_desc": "2.5.2",
            "time_connected": "16:49:01.795000"
        },
        {
            "_created": "00:00:00.000000",
            "_etag": "e8ff778368901540349b2a9625893b1b4763b362",
            "_id": "59854e41e14ebffa9f4f4e80",
            "_updated": "00:00:00.000000",
            "dp_desc": "None",
```

```
            "dpid": 2,
            "hw_desc": "Open vSwitch",
            "ip_address": "172.16.0.9",
            "mfr_desc": "Nicira, Inc.",
            "port": 34090,
            "serial_num": "None",
            "sw_desc": "2.5.2",
            "time_connected": "16:49:05.706000"
        }
    ],
    "_meta": {
        "max_results": 25,
        "page": 1,
        "total": 2
    }
}
```

Switch Count

The Switch Count API is a read-only count of all switches currently connected to controller.

Example manual invocation of the API:

```
curl http://localhost:8081/v1/infrastructure/switches/stats/connected_switches |␣
↪python -m json.tool
```

Example response:

```
{
    "connected_switches": 2
}
```

Internal APIs

No internal APIs exist yet. They are planned to implement connectivity between the API instance and the main nmeta code for interaction into non-database components of nmeta.

CHAPTER 8

Extend Nmeta

Custom Classifiers

Nmeta supports the creation of custom classifiers to extend classification, leveraging any network metadata. See the configure chapter for how to reference a custom classifier from main_policy.yaml.

Custom classifiers have access to the flow and identity abstractions (see develop chapter)

Develop

Want to develop the nmeta code and contribute to the codebase? Great! Read on for documentation on how the code is structured and some of the principles.

Code Structure

Nmeta runs in 3 separate processes.

- The *Ryu process* runs within the context of the Ryu OpenFlow controller.

- The *MongoDB process* is a MongoDB database

- The *api_external* process runs the external REST API
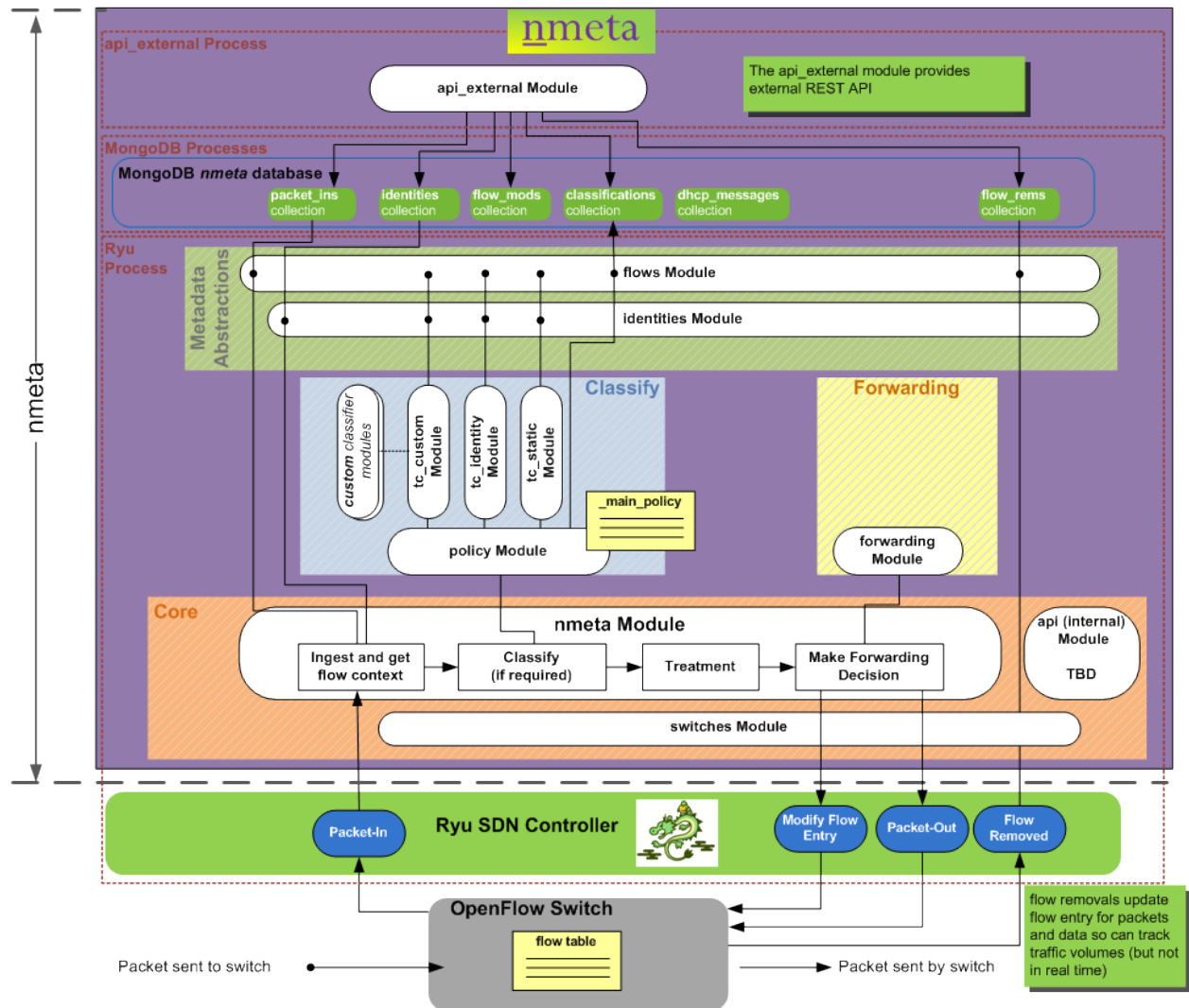
Data Structures

Nmeta uses various data structures to store network metadata related to participants and flows (conversations).

High level abstractions of participants and flows abstract the details of the various MongoDB collections.

Information Abstractions

Flows Abstraction

The flows object provides an abstraction of flows (conversations) that have been seen on the network. Flow metrics are in the context of the flow that the last packet-in ingested packet belonged to. The packet context is likewise that of the packet from that event.

Flows are a **network-layer view of conversations**. Points to consider:
- The hash is an indexed bi-directionally-unique value, derived from IP-value ordered 5-tuple
- A hash could be reused in real life, i.e. due to source-port exhaustion (corner case, but should be handled)
- The same packet may be sent to the controller by different switches (duplication)
- A packet may be retransmitted (duplication)

The **flow_hash** uniquely identifies a flow. It is an indexed bi-directionally-unique value, derived from a value ordered 5-tuple

The **packet_hash** is used for packet de-duplication. It is an indexed uni-directionally packet identifier, derived from ip_src, ip_dst, proto, tp_src, tp_dst, tp_seq_src, tp_seq_dst

'flow'

Packet-In **Flow**

Identification

flow_hash

Flow **Behaviour**

packet_count()

client()

server()

packet_directions()

packet_sizes()

max_packet_size()

max_interpacket_interval()

min_interpacket_interval()

Flow **Classification**

'classification'

classified

classification_tag

classification_time

actions{}

commit()

Ingest a Packet-In Event

ingest_packet(dpid, in_port, pkt, timestamp)

Packet-In **Packet**

Packet-In event creates packet-ins col record

'packet'

Identification

flow_hash

packet_hash

Layer-2

eth_src

eth_dst

eth_type

Layer-3

ip_src

ip_dst

proto

Layer-4

tp_src

tp_dst

tp_flags

tp_seq_src

tp_seq_dst

Layer-7

payload

Environmental

dpid

in_port

timestamp

length

Values stored in packet-ins MongoDB collection record

Calculated Values

packet_direction()

TCP-specific Flags

tcp_fin

tcp_syn

tcp_rst

tcp_psh

tcp_ack

tcp_urg

tcp_ece

tcp_cwr

Has been classified (whether it matched or not)

Write record to database

Classifiers can make use of the flows object to gain easy access to features of the current flow.

Identities Abstraction

The identities object provides an abstraction for participants (identities) that are known to nmeta. Classifiers can use the identities object to look up the identity information of participants.
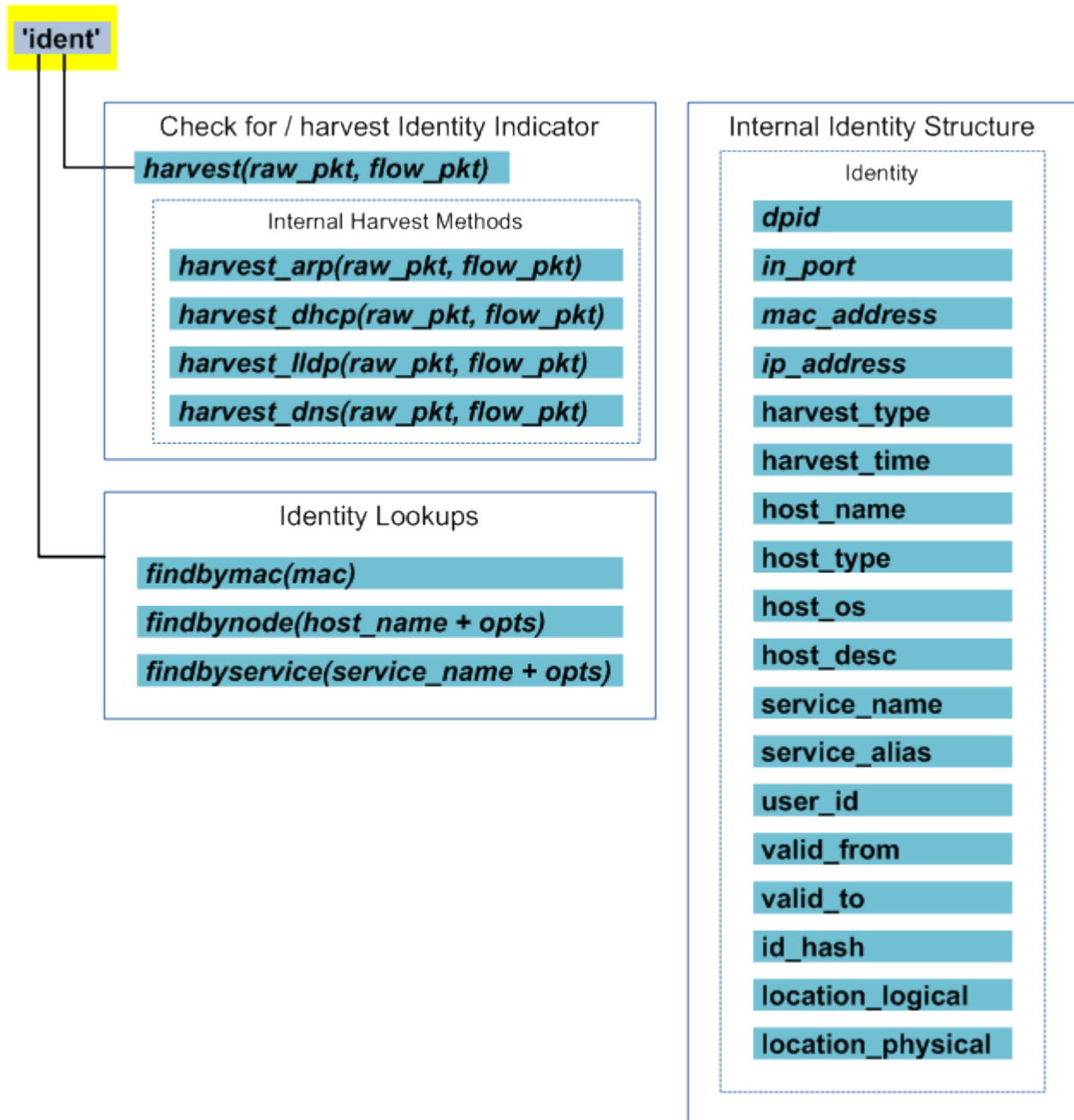
Database Collections

Nmeta uses capped MongoDB database collections to obviate the need to maintain size by pruning old entries.

packet-ins

Documents in the **packet_ins** MongoDB collection are **immutable**, so that they can be stored in a **capped collection**, which is fast and automatically overwrites oldest documents when it reaches max size in bytes. Documents in capped collections cannot be increased in size, so it is easiest to treat them as immutable. Their data can be mined for flow metadata, packet data and performance metrics.

'packet_ins'

- _id
- flow_hash
- packet_hash
- dpid
- in_port
- timestamp
- length
- eth_src
- eth_dst
- eth_type
- ip_src
- ip_dst
- proto
- tp_src
- tp_dst
- tp_flags
- tp_seq_src
- tp_seq_dst
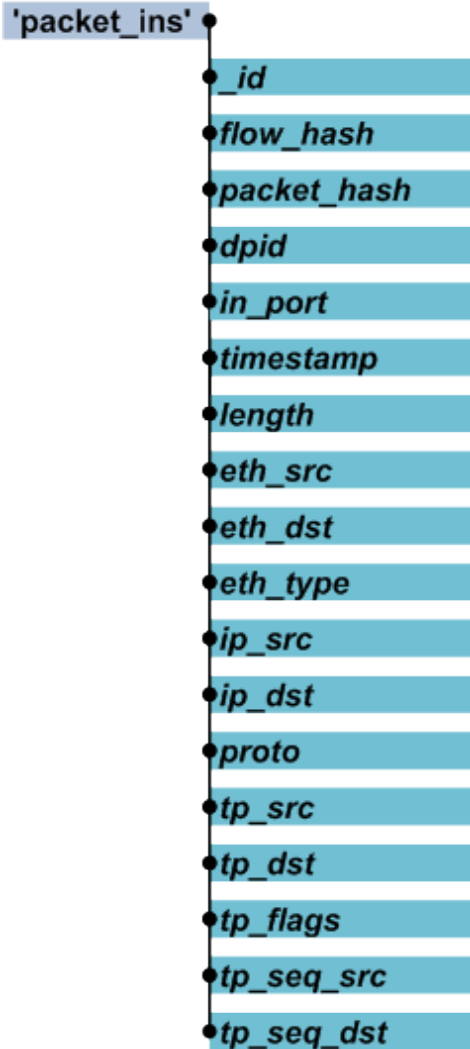
pi_time

The pi_time database collection stores data on how long nmeta took to process individual packet-in events, and what type of outcome nmeta decided upon for the packet.

Documents in the **pi_time** MongoDB collection are **immutable**, so that they can be stored in a **capped collection**, which is fast and automatically overwrites oldest documents when it reaches max size in bytes. Documents in capped collections cannot be increased in size, so it is easiest to treat them as immutable.

'pi_time'

_id

pi_delta

outcome

timestamp

pi_delta is the time in seconds nmeta took to process this packet_in event

outcome is one of:
- drop_same_port
- drop_reserved_mac
- drop_action (dropped by policy)
- packet_out_flooded
- packet_out

timestamp is the time when this record was created
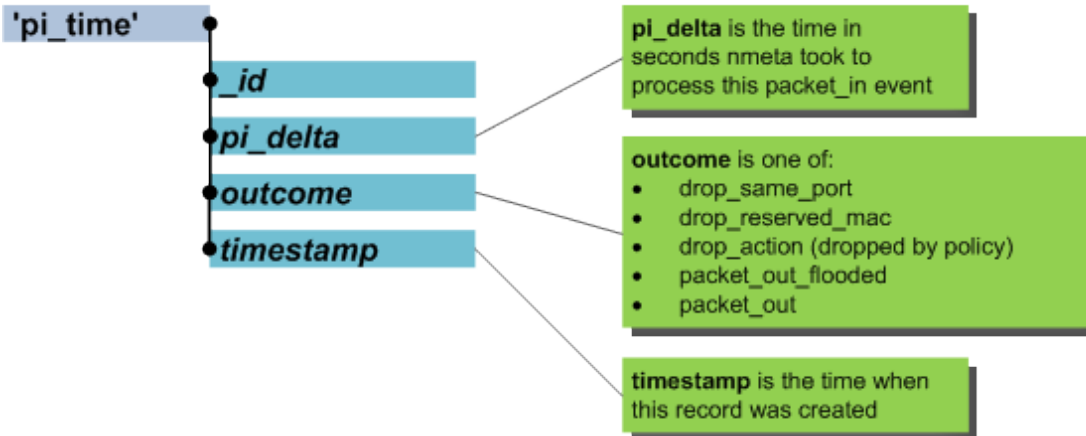
classifications

The **classifications** database collection is holds a record of traffic classifications made by nmeta, along with actions to take. Points to consider:
- The *classified* flag says whether or not more packets need to be seen to make a determination

'classifications'

_id

flow_hash

classified()

classification_type

classification_tag

classification_time

actions()

identities

The **identities** database collection is a **view of network participants**. Points to consider:
 - Assumed that a participant always has an IP address (beware: this may not always hold true)
 - Any valid IP version (i.e. IPv4 or IPv6)
 - An IP address can be shared by multiple participants concurrently (examples: participants behind a proxy server or sharing a server)
 - An IP address may be dedicated to a participant, but only for a period of time (example: DHCP-assigned address on Wi-Fi)

'identities'
- _id
- dpid
- in_port
- mac_address
- ip_address
- harvest_type
- harvest_time
- host_name
- host_type
- host_os
- host_desc
- service_name
- service_alias
- user_id
- valid_from
- valid_to
- id_hash
- location_logical
- location_physical

Valid **havest_type** values are:
    ARP
    DHCP
    LLDP
    DNS_A
    DNS_CNAME

The **id_hash** is used for identity de-duplication. It is derived from mac_address, ip_address, harvest_type, host_name, service_name, user_id

The **location_logical** key is a user-defined string derived through policy from the source dpid/port

The **location_physical** key is a placeholder for a future description of the identity physical location based on inputs such as RF triangulation

flow_mods

Documents in the **flow_mods** MongoDB collection are **immutable**, so that they can be stored in a **capped collection**, which is fast and automatically overwrites oldest documents when it reaches max size in bytes. Documents in capped collections cannot be increased in size, so it is easiest to treat them as immutable. Their data can be mined for flow metadata, packet data and performance metrics.

'flow_mods'

_id

flow_hash

dpid

timestamp

suppress_type

Valid **suppress_type** values are:
'suppress': forwards the flow on the switch such that it doesn't send more packets to the controller

'drop': drops all packets in this flow at the switch

standdown

Reduce risk of overloading switch with duplicate suppression events. If 1 then already suppressed recently so don't mod.

match_type

Match type set by switches module (ignore|single|dual). Ignore means no mod, dual had forward and reverse mods

forward_cookie

forward_match

reverse_cookie

Unique value per flow mod, set by nmeta

reverse_match

matches are dict objects of the flow match parameters. Reverse only used in match_type of dual (TCP flows)

client_ip

IP address of client in flow (first seen) or 0 if not known

flow_rems

The **flow_rems** database collection is a **view of flows removed from switches**.

'flow_rems'
- _id
- dpid
- removal_time
- cookie
- priority
- reason
- table_id
- duration_sec
- idle_timeout
- hard_timeout
- packet_count
- byte_count
- eth_A
- eth_B
- eth_type
- ip_A
- ip_B
- ip_proto
- tp_A
- tp_B
- flow_hash
- direction

Session direction (forward|reverse).
Calculated from cookie.
Defaults to forward

dhcp_v4

The **dhcp_messages** database collection holds state for IPv4 DHCP transactions

'dhcp_messages'

_id
dpid
in_port
ingest_time
eth_src
eth_dst
ip_src
ip_dst
tp_src
tp_dst
transaction_id
message_type
host_name
ip_assigned
ip_dhcp_server
lease_time

transaction_id — Used to link multiple DHCP events

host_name — host name as defined by DHCP client

ip_assigned — IP address assigned by DHCP (aka YIADDR)

ip_dhcp_server — IP address of the DHCP server

Logging

Logging is configured separately for syslog and to the console, and levels are configurable per Python module. The log format is also customisable.

Logging configuration is controlled by the system configuration YAML file.

Logging settings are configured separately for *console* and *syslog* logging.

By default, logging levels are set to INFO.

Supported logging levels are:

- CRITICAL
- ERROR
- WARNING

- INFO

- DEBUG

To change the default logging levels, create a user configuration YAML file (if it doesn't already exist) as the following filename:

```
~/nmeta/nmeta/config/user/config.yaml
```

Override specific settings from the default configuration file from the directory below.

Example:

```
# Set nmeta.py console logging to DEBUG level:
nmeta_logging_level_c: DEBUG
```

Code Documentation

nmeta module

This is the main module of the nmeta suite running on top of Ryu SDN controller to provide network identity and flow (traffic classification) metadata . Do not use this code for production deployments - it is proof of concept code and carries no warrantee whatsoever. You have been warned.

class nmeta.**NMeta**(**args*, ***kwargs*)

> Bases: ryu.base.app_manager.RyuApp, baseclass.BaseClass
>
> This is the main class used to run nmeta
>
> **OFP_VERSIONS = [4]**
>
> **switch_connection_handler**(*event*)
> > A switch has connected to the SDN controller. We need to do some tasks to set the switch up properly such as setting it's config for fragment handling and table miss packet length and requesting the switch description
>
> **desc_stats_reply_handler**(*event*)
> > Receive a reply from a switch to a description statistics request
>
> **switch_down_handler**(*event*)
> > OpenFlow state has gone down for a given DPID
>
> **packet_in**(*event*)
> > This method is called for every Packet-In event from a Switch. We receive a copy of the Packet-In event, pass it to the traffic classification area for analysis, work out the forwarding, update flow metadata, then add a flow entry to the switch (when appropriate) to suppress receiving further packets on this flow. Finally, we send the packet out the switch port(s) via a Packet-Out message, with appropriate QoS queue set.
>
> **flow_removed_handler**(*event*)
> > A switch has sent an event to us because it has removed a flow from a flow table
>
> **error_msg_handler**(*event*)
> > A switch has sent us an error event

> **_port_status_handler**(*event*)
>> Switch Port Status event

class nmeta.**PITelemetry**(*pi_start_time*, *event*, *logger*, *pi_time_col*)
> Bases: object

> Telemetry data for a single Packet-In (PI) event

> **record_outcome**(*outcome*)
>> Calculate the elapsed time for processing this packet-in event and record to the pi_time database collection. Also record the outcome for the packet, one of: - drop_same_port - drop_reserved_mac - drop_action - packet_out_flooded - packet_out Additionally, record time taken queueing event in Ryu (if available).

nmeta.**ipv4_text_to_int**(*ip_text*)
> Takes an IP address string and translates it to an unsigned integer

policy module

This module is part of the nmeta suite running on top of Ryu SDN controller. It provides a policy class as an interface to policy configuration and classification of packets against policy.

See Policy class docstring for more information.

policy.**validate**(*logger*, *data*, *schema*, *where*)
> Generic validation of a data structure against schema using Voluptuous data validation library Parameters:

>> •logger: valid logger reference

>> •data: structure to validate

>> •schema: a valid Voluptuous schema

>> •where: string for debugging purposes to identity the policy location

policy.**validate_port_set_list**(*logger*, *port_set_list*, *policy*)
> Validate that a list of dictionaries [{'port_set': str}] reference valid port_sets. Return Boolean 1 if good otherwise exit with exception

policy.**validate_location**(*logger*, *location*, *policy*)
> Validator for location compliance (i.e. check that the supplied location string exists as a location defined in policy) Return Boolean True if good, otherwise exit with exception

policy.**validate_type**(*type*, *value*, *msg*)
> Used for Voluptuous schema validation. Check a value is correct type, otherwise raise Invalid exception, including elaborated version of msg

policy.**transform_ports**(*ports*)
> Passed a ports specification and return a list of port numbers for easy searching. Example: Ports specification "1-3,5,66" becomes list [1,2,3,5,66]

policy.**validate_ports**(*ports*)
> Custom Voluptuous validator for a list of ports. Example good ports specification:

>> 1-3,5,66

> Will raise Voluptuous Invalid exception if types or ranges are not correct

policy.**validate_time_of_day**(*time_of_day*)
> Custom Voluptuous validator for time of day compliance. Returns original time of day if compliant, otherwise raises Voluptuous Invalid exception

policy.**validate_macaddress**(*mac_addr*)
: Custom Voluptuous validator for MAC address compliance. Returns original MAC address if compliant, otherwise raises Voluptuous Invalid exception

policy.**validate_macaddress_OLD**(*mac_addr*)
: Custom Voluptuous validator for MAC address compliance. Returns original MAC address if compliant, otherwise raises Voluptuous Invalid exception

policy.**validate_ip_space**(*ip_addr*)
: Custom Voluptuous validator for IP address compliance. Can be IPv4 or IPv6 and can be range or have CIDR mask. Returns original IP address if compliant, otherwise raises Voluptuous Invalid exception

policy.**validate_ethertype**(*ethertype*)
: Custom Voluptuous validator for ethertype compliance. Can be in hex (starting with 0x) or decimal. Returns ethertype if compliant, otherwise raises Voluptuous Invalid exception

class policy.**Policy**(*config*, *pol_dir_default='config'*, *pol_dir_user='config/user'*, *pol_filename='main_policy.yaml'*)
: Bases: baseclass.BaseClass

    This policy class serves 4 main purposes: - Ingest policy (main_policy.yaml) from file - Validate correctness of policy against schema - Classify packets against policy, passing through to static,

        identity and custom classifiers, as required

        •Other methods and functions to check various parameters against policy

    Note: Class definitions are not nested as not considered Pythonic

    Main Methods and Variables: - check_policy(flow, ident) # Check a packet against policy - qos(qos_treatment) # Map qos_treatment string to queue number - main_policy # main policy YAML object. Read-only,

        no verbs. Use methods instead where possible.

    TC Methods and Variables: - tc_rules.rules_list # List of TC rules - tc_rules.custom_classifiers # dedup list of custom classifier names

    check_policy(*flow*, *ident*)
    : Passed a flows object, set in context of current packet-in event, and an identities object. Check if packet matches against any policy rules and if it does, update the classifications portion of the flows object to reflect details of the classification.

    qos(*qos_treatment*)
    : Passed a QoS treatment string and return the relevant QoS queue number to use, otherwise 0. Works by lookup on qos_treatment section of main_policy

class policy.**TCRules**(*policy*)
: Bases: object

    An object that represents the tc_rules root branch of the main policy

class policy.**TCRule**(*tc_rules*, *policy*, *idx*)
: Bases: object

    An object that represents a single traffic classification (TC) rule.

    check_tc_rule(*flow*, *ident*)
    : Passed Packet and Identity class objects. Check to see if packet matches conditions as per the TC rule. Return a TCRuleResult object

class policy.**TCRuleResult**(*rule_actions*)
: Bases: object

An object that represents a traffic classification result, including any decision collateral on matches and actions. Use __dict__ to dump to data to dictionary

accumulate(*condition_result*)
　　Passed a TCConditionResult object and accumulate values into our object

add_rule_actions()
　　Add rule actions from policy to the actions of this class

class `policy.`**`TCCondition`**(*tc_rules*, *policy*, *policy_snippet*)
　　Bases: `object`

　　An object that represents a single traffic classification (TC) rule condition from a conditions list (contains a match type and a list of one or more classifiers)

　　check_tc_condition(*flow*, *ident*)
　　　　Passed a Flow and Identity class objects. Check to see if flow.packet matches condition (a set of classifiers) as per the match type. Return a TCConditionResult object with match information.

class `policy.`**`TCConditionResult`**
　　Bases: `object`

　　An object that represents a traffic classification condition result. Custom classifiers can return additional parameters beyond a Boolean match, so cater for these too. Use __dict__ to dump to data to dictionary

　　accumulate(*classifier_result*)
　　　　Passed a TCClassifierResult object and accumulate values into our object

class `policy.`**`TCClassifierResult`**(*policy_attr*, *policy_value*)
　　Bases: `object`

　　An object that represents a traffic classification classifier result. Custom classifiers can return additional parameters beyond a Boolean match, so cater for these too. Use __dict__ to dump to data to dictionary

class `policy.`**`QoSTreatment`**(*policy*)
　　Bases: `object`

　　An object that represents the qos_treatment root branch of the main policy

class `policy.`**`PortSets`**(*policy*)
　　Bases: `object`

　　An object that represents the port_sets root branch of the main policy

　　get_port_set(*dpid*, *port*, *vlan_id=0*)
　　　　Check if supplied dpid/port/vlan_id is member of a port set and if so, return the port_set name. If no match return empty string.

class `policy.`**`PortSet`**(*policy*, *idx*)
　　Bases: `object`

　　An object that represents a single port set

　　is_member(*dpid*, *port*, *vlan_id=0*)
　　　　Check to see supplied dpid/port/vlan_id is member of this port set. Returns a Boolean

class `policy.`**`Locations`**(*policy*)
　　Bases: `object`

　　An object that represents the locations root branch of the main policy

　　get_location(*dpid*, *port*)
　　　　Passed a DPID and port and return a logical location name, as per policy configuration.

class `policy.**Location**`(*policy*, *idx*)

> Bases: `object`
>
> An object that represents a single location
>
> **check**(*dpid*, *port*)
>
> > Check a dpid/port to see if it is part of this location and if so return the string name of the location otherwise return empty string

tc_static module

This module is part of the nmeta suite running on top of Ryu SDN controller to provide network identity and flow (traffic classification) metadata

class `tc_static.**StaticInspect**`(*config*, *policy*)

> Bases: `baseclass.BaseClass`
>
> This class provides methods to check static traffic classification (TC) classifier matches
>
> **check_static**(*classifier_result*, *pkt*)
>
> > Passed TCClassifierResult and Flow.Packet class objects Update the classifier_result match with boolean of result of match checks
>
> **is_valid_macaddress**(*value_to_check*)
>
> > Passed a prospective MAC address and check that it is valid. Return 1 for is valid IP address and 0 for not valid
>
> **is_valid_ethertype**(*value_to_check*)
>
> > Passed a prospective EtherType and check that it is valid. Can be hex (0x*) or decimal Return 1 for is valid IP address and 0 for not valid
>
> **is_valid_ip_space**(*value_to_check*)
>
> > Passed a prospective IP address and check that it is valid. Can be IPv4 or IPv6 and can be range or have CIDR mask Return 1 for is valid IP address and 0 for not valid
>
> **is_valid_transport_port**(*value_to_check*)
>
> > Passed a prospective TCP or UDP port number and check that it is an integer in the correct range. Return 1 for is valid port number and 0 for not valid port number
>
> **is_match_time_of_day**(*time_of_day_range*, *time_now=datetime.time(21, 47, 34, 926922)*)
>
> > Passed a time of day range (format HH:MM-HH:MM) and check to see if the current time is in that range. Return True if time is in range, otherwise False
>
> **is_match_macaddress**(*value_to_check1*, *value_to_check2*)
>
> > Passed a two prospective MAC addresses and check to see if they are the same address. Return 1 for both the same MAC address and 0 for different
>
> **is_match_ethertype**(*value_to_check1*, *value_to_check2*)
>
> > Passed a two prospective EtherTypes and check to see if they are the same. Return 1 for both the same EtherType and 0 for different Values can be hex or decimal and are 2 bytes in length
>
> **is_match_ip_space**(*ip_addr*, *ip_space*)
>
> > Passed an IP address and an IP address space and check if the IP address belongs to the IP address space. If it does return 1 otherwise return 0

tc_identity module

This module is part of the nmeta suite running on top of Ryu SDN controller to provide network identity and flow (traffic classification) metadata

class tc_identity.**IdentityInspect**(*config*)

    Bases: baseclass.BaseClass

    This class is instantiated by policy.py and provides methods to ingest identity updates and query identities

    check_identity(*classifier_result*, *pkt*, *ident*)

        Checks if a given packet matches a given identity match rule. Passed TCClassifierResult, Flow.Packet and Identities class objects and update the classifier_result match based on whether or not either of the packet IP addresses matches the identity attribute/value. Uses methods of the Identities class to work this out

    check_lldp(*host_name*, *pkt*, *ident*, *is_regex=False*)

        Passed a hostname, flows packet object, an instance of the identities class and a regex boolean (if true, hostname is treated as regex). Return True or False based on whether or not the packet has a source or destination IP address that matches the IP address registered to the given hostname via LLDP harvest (if one even exists). Uses methods of the Identities class to work this out. Returns boolean

    check_dhcp(*host_name*, *pkt*, *ident*, *is_regex=False*)

        Passed a hostname, flows packet object, an instance of the identities class and a regex boolean (if true, hostname is treated as regex). Return True or False based on whether or not the packet has a source or destination IP address that matches the IP address registered to the given hostname via DHCP harvest (if one even exists). Uses methods of the Identities class to work this out. Returns boolean

    check_dns(*dns_name*, *pkt*, *ident*, *is_regex=False*)

        Passed a DNS name, flows packet object, an instance of the identities class and a regex boolean (if true, DNS name is treated as regex). Return True or False based on whether or not the packet has a source or destination IP address that has been resolved from the DNS name. Uses methods of the Identities class to work this out. Returns boolean

tc_custom module

This module is part of the nmeta suite running on top of Ryu SDN controller to provide network identity and flow (traffic classification) metadata

class tc_custom.**CustomInspect**(*config*)

    Bases: baseclass.BaseClass

    This class is instantiated by policy.py and provides methods to run custom traffic classification modules

    check_custom(*classifier_result*, *flow*, *ident*)

        Passed TCClassifierResult, Flow.Packet and Identities class objects. Call the named custom classifier with these values so that it can update the classifier_result match as appropriate.

    instantiate_classifiers(*custom_list*)

        Dynamically import and instantiate classes for any custom classifiers specified in the controller nmeta2 main_policy.yaml

        Passed a deduplicated list of custom classifier names (without .py) to load.

        Classifier modules live in the 'custom_classifiers' subdirectory

api_external module

The api_external module is part of the nmeta suite, but is run separately

This module runs a class and methods for an API that exposes an interface into nmeta MongoDB collections.

It leverages the Eve Python REST API Framework

class `api_external.`**`ExternalAPI`**(*config*)

> Bases: `baseclass.BaseClass`
>
> This class provides methods for the External API
>
> **class FlowUI**
>
> > Bases: `object`
> >
> > An object that represents a flow record to be sent in response to the WebUI. Features:
> >
> > > • Flow direction normalised to direction of first packet in flow
> > >
> > > • Src and Dst are IP or Layer 2 to optimise screen space
> > >
> > > • Extra data included for hover-over tips
> >
> > Note that there should not be any display-specific data (i.e. don't send any HTML, leave this to the client code)
> >
> > **`response`**()
> >
> > > Return a dictionary object of flow parameters for sending in response
>
> `ExternalAPI.`**`run`**()
>
> > Run the External API instance
> >
> > Note that API definitions are from previously imported files from api_definitions subdirectory
>
> `ExternalAPI.`**`response_pi_rate`**(*items*)
>
> > Update the response with the packet_in rate. Hooked from on_fetched_resource_pi_rate
> >
> > Returns key/values for packet-in processing time in API response: - timestamp - pi_rate
>
> `ExternalAPI.`**`response_pi_time`**(*items*)
>
> > Update the response with the packet_time min, avg and max. Hooked from on_fetched_resource_pi_time
> >
> > Returns key/values for packet-in processing time in API response: - timestamp - ryu_time_max - ryu_time_min - ryu_time_avg - ryu_time_period - ryu_time_records - pi_time_max - pi_time_min - pi_time_avg - pi_time_period - pi_time_records
> >
> > If no data found within time period then returns without key/values
>
> `ExternalAPI.`**`response_controller_summary`**(*items*)
>
> > Update the response with the packet_in rate, packet processing time stats
> >
> > Hooked from on_fetched_resource_controller_summary
> >
> > Rounds seconds results
>
> `ExternalAPI.`**`response_identities_ui`**(*items*)
>
> > **Populate the response with identities that are filtered:**
> >
> > > • Reverse sort by harvest time
> > >
> > > • Deduplicate by id_hash, only returning most recent per id_hash
> > >
> > > • Includes possibly stale records
> > >
> > > • Check DNS A records to see if they are from a CNAME

Hooked from on_fetched_resource_<name>

ExternalAPI.**response_flows_removed_stats_count**(*items*)
Return count of removed flows collection

ExternalAPI.**response_flows_removed_src_bytes_sent**(*items*)
Returns removed flow bytes sent by session source IP (deduplicated for flows crossing multiple switches), enriched with identity metadata.

ExternalAPI.**response_flows_removed_src_bytes_received**(*items*)
Returns removed flow bytes received by session source IP (deduplicated for flows crossing multiple switches), enriched with identity metadata.

ExternalAPI.**response_flows_removed_dst_bytes_sent**(*items*)
Returns removed flow bytes sent by session destination IP (deduplicated for flows crossing multiple switches), enriched with identity metadata.

ExternalAPI.**response_flows_removed_dst_bytes_received**(*items*)
Returns removed flow bytes received by session destination IP (dedup for flows crossing multiple switches), enriched with identity metadata.

ExternalAPI.**response_flows_ui**(*items*)

Populate the response with flow entries that are filtered:

- Reverse sort by initial ingest time

- Deduplicate by flow_hash, only returning most recent per flow_hash

- Enrich with TBD

Hooked from on_fetched_resource_<name>

ExternalAPI.**response_switches_count**(*items*)
Populate the response with number of connected switches.

ExternalAPI.**flow_match**(*flow*, *flows_filterlogicselector*, *flows_filtertypeselector*, *filter_string*)
Passed an instance of FlowUI class, a logic selector, filter type and filter string.

Return a boolean on whether or not that theres a match.

ExternalAPI.**flow_augment_record**(*record*)
Passed a record of a single flow from the packet_ins database collection.

Create FlowUI class instance, add in known data and augment with identity data. Logic is specific to the webUI user experience.

Return the FlowUI class instance

ExternalAPI.**get_flow_data_xfer**(*record*)
Passed a record of a single flow from the packet_ins database collection.

Enrich this by looking up data transfer stats (which may not exist) in flow_rems database collection, and return dictionary of the values.

Note that the data sent (tx) and received (rx) records will have different flow hashes.

ExternalAPI.**get_classification**(*flow_hash*)
Passed flow_hash and return a dictionary of a classification object for the flow_hash (if found), otherwise a dictionary of an empty classification object.

ExternalAPI.**flow_normalise_direction**(*record*)
Passed a dictionary of an flow record and return a similar dictionary that has sources and destinations normalised to the direction of the first observed packet in the flow

ExternalAPI.**get_flow_client_ip**(*flow_hash*)
: Find the IP that is the originator of a flow searching forward by flow_hash

    Finds first packet seen for the flow_hash within the time limit and returns the source IP, otherwise 0,

ExternalAPI.**get_id**(*ip_addr*)
: Passed an IP address. Look this up for matching identity metadata and return a string that contains either the original IP address or an identity string

ExternalAPI.**get_dns_ip**(*service_name*)
: Use this to get an IP address for a DNS lookup that returned a CNAME Passed a DNS CNAME and look this up in identities collection to see if there is a DNS A record, and if so return the IP address, otherwise return an empty string.

ExternalAPI.**get_host_by_ip**(*ip_addr*)
: Passed an IP address. Look this up in the identities db collection and return a host name if present, otherwise an empty string

ExternalAPI.**get_location_by_mac**(*mac_addr*)
: Passed a MAC address. Look this up in the identities db collection and return a source logical location if present, otherwise an empty string

ExternalAPI.**get_service_by_ip**(*ip_addr*, *alias=1*)
: Passed an IP address. Look this up in the identities db collection and return a service name if present, otherwise an empty string.

    If alias is set, do additional lookup on success to see if service name is an alias for another name, and if so return that.

ExternalAPI.**get_pi_rate**(*test=0*)
: Calculate packet-in rate by querying packet_ins database collection.

    Setting test=1 returns database query execution statistics

ExternalAPI.**get_pi_time**()
: Calculate packet processing time statistics by querying the pi_time database collection.

api_external.**enumerate_eth_type**(*eth_type*)
: Passed an eth_type (in decimal) and return an enumerated version, or if not found, return the original value. Example, pass this function value 2054 and it return will be 'ARP'

api_external.**hovertext_eth_type**(*eth_type*)
: Passed an eth_type (decimal, not enumerated) and return it wrapped in extra text to convey context

api_external.**enumerate_ip_proto**(*ip_proto*)
: Passed an IP protocol number (in decimal) and return an enumerated version, or if not found, return the original value. Example, pass this function value 6 and it return will be 'TCP'

api_external.**hovertext_ip_proto**(*ip_proto*)
: Passed an IP protocol number (decimal, not enumerated) and return it wrapped in extra text to convey context

api_external.**hovertext_ip_addr**(*ip_addr*)
: Passed an IP address and return it wrapped in extra text to convey context

config module

The config module is part of the nmeta suite.

It represents nmeta configuration data.

It loads configuration from file, validates keys and provides access to values

It expects a file called "config.yaml" to be in the config subdirectory, containing properly formed YAML

class config.**Config**(*dir_default='config'*, *dir_user='config/user'*, *config_filename='config.yaml'*)
　　Bases: baseclass.BaseClass

　　This class is instantiated by nmeta.py and provides methods to ingest the configuration file and provides access to the keys/values that it contains. Config file is in YAML in config subdirectory and is called 'config.yaml'

　　ingest_config_default(*config_filename*, *dir_default*)
　　　　Ingest default config file

　　ingest_config_user(*config_filename*, *dir_user*)
　　　　Ingest user config file that overrides values set in the default config file.

　　ingest_config_file(*fullpath*)
　　　　Passed full path to a YAML-formatted config file and ingest into a dictionary

　　get_value(*config_key*)
　　　　Passed a key and see if it exists in the config YAML. If it does then return the value, if not return 0

　　inherit_logging(*config*)
　　　　Call base class method to set up logging properly for this class now that it is running

flows module

The flows module is part of the nmeta suite

It provides an abstraction for conversations (flows), using a MongoDB database for storage and data retention maintenance.

Flows are identified via an indexed bi-directionally-unique hash value, derived from IP-value-ordered 5-tuple (source and destination IP addresses, IP protocol and transport source and destination port numbers).

Ingesting a packet puts the flows object into the context of the packet that flow belongs to, and updates the database object for that flow with information from the current packet.

There are various methods (see class docstring) that provide views into the state of the flow.

class flows.**Flow**(*config*)
　　Bases: baseclass.BaseClass

　　An object that represents a flow that we are classifying

　　Intended to provide an abstraction of a flow that classifiers can use to make determinations without having to understand implementations such as database lookups etc.

　　Be aware that this module is not very mature yet. It does not cover some basic corner cases such as packet retransmissions and out of order or missing packets.

　　Read a packet_in event into flows (assumes class instantiated as an object called 'flow'):

　　　　flow.ingest_packet(dpid, in_port, pkt, timestamp)

　　Variables available for Classifiers (assumes class instantiated as an object called 'flow'):

　　　　Variables for the current packet:

　　　　flow.packet.flow_hash The hash of the 5-tuple of the current packet

　　　　flow.packet.packet_hash The hash of the current packet used for deduplication. It is an indexed uni-directionally packet identifier, derived from ip_src, ip_dst, proto, tp_src, tp_dst, tp_seq_src, tp_seq_dst

flow.packet.dpid The DPID that the current packet was received from via a Packet-In message

flow.packet.in_port The switch port that the current packet was received on before being sent to the controller

flow.packet.timestamp The time in datetime format that the current packet was received at the controller

flow.packet.length Length in bytes of the current packet on wire

flow.packet.eth_src Ethernet source MAC address of current packet

flow.packet.eth_dst Ethernet destination MAC address of current packet

flow.packet.eth_type Ethertype of current packet in decimal

flow.packet.ip_src IP source address of current packet

flow.packet.ip_dst IP destination address of current packet

flow.packet.proto IP protocol number of current packet

flow.packet.tp_src Source transport-layer port number of current packet

flow.packet.tp_dst Destination transport-layer port number of current packet

flow.packet.tp_flags Transport-layer flags of the current packet

flow.packet.tp_seq_src Source transport-layer sequence number (where existing) of current packet

flow.packet.tp_seq_dst Destination transport-layer sequence number (where existing) of current packet

flow.packet.payload Payload data of current packet

flow.packet.tcp_fin() True if TCP FIN flag is set in the current packet

flow.packet.tcp_syn() True if TCP SYN flag is set in the current packet

flow.packet.tcp_rst() True if TCP RST flag is set in the current packet

flow.packet.tcp_psh() True if TCP PSH flag is set in the current packet

flow.packet.tcp_ack() True if TCP ACK flag is set in the current packet

flow.packet.tcp_urg() True if TCP URG flag is set in the current packet

flow.packet.tcp_ece() True if TCP ECE flag is set in the current packet

flow.packet.tcp_cwr() True if TCP CWR flag is set in the current packet

flow.packet_direction() c2s (client to server) or s2c direction of current packet

Variables for the whole flow:

flow.packet_count() Unique packets registered for the flow

flow.client() The IP that is the originator of the flow

flow.server() The IP that is the destination of the flow

flow.packet_directions() List of packet directions for the flow

flow.packet_sizes() List of packet sizes (lengths, in bytes) for the flow

flow.max_packet_size() Size of largest packet in the flow in bytes

flow.max_interpacket_interval() Maximum directional time difference between packets

flow.min_interpacket_interval() Minimum directional time difference between packets

The Flow class also includes the record_removal method that records a flow removal message from a switch to database

Challenges (not handled - yet):

- duplicate packets due to retransmissions

- IP fragments

- Flow reuse - TCP source port reused

class **Packet**

Bases: `object`

An object that represents the current packet

dbdict()

Return a dictionary object of metadata parameters of current packet (excludes payload), for storing in database

tcp_fin()

Does the current packet have the TCP FIN flag set?

tcp_syn()

Does the current packet have the TCP SYN flag set?

tcp_rst()

Does the current packet have the TCP RST flag set?

tcp_psh()

Does the current packet have the TCP PSH flag set?

tcp_ack()

Does the current packet have the TCP ACK flag set?

tcp_urg()

Does the current packet have the TCP URG flag set?

tcp_ece()

Does the current packet have the TCP ECE flag set?

tcp_cwr()

Does the current packet have the TCP CWR flag set?

class Flow.**Classification**(*flow_hash*, *clsfn*, *time_limit*, *logger*)

Bases: `object`

An object that represents an individual traffic classification

test_query()

Return database query execution statistics

dbdict()

Return a dictionary object of traffic classification parameters for storing in the database

commit()

Record current state of flow classification into MongoDB classifications collection.

class Flow.**RemovedFlow**(*logger*, *flow_rems*, *msg*, *offset*)

Bases: `object`

An object that represents an individual removed flow. This is a flow that a switch has informed us it has removed from its flow table because of an idle timeout

dbdict()
    Return a dictionary object of parameters from the removed flow for storing in the flow_rems database collection

commit()
    Record removed flow into MongoDB flow_rems collection.

Flow.**record_removal**(*msg*)
    Record an idle-timeout flow removal message. Passed a Ryu message object for the flow removal. Record entry in the flow_rems database collection

Flow.**ingest_packet**(*dpid*, *in_port*, *packet*, *timestamp*)
    Ingest a packet into the packet_ins collection and put the flow object into the context of the packet. Note that timestamp MUST be in datetime format

Flow.**packet_count**(*test=0*)
    Return the number of packets in the flow (counting packets in both directions). This method deduplicates for where the same packet is received from multiple switches, by filtering to the DPID from which the first packet-in for the flow was received (could be wrong in obscure corner cases).

    Works by retrieving packets from packet_ins database with current packet flow_hash and within flow reuse time limit.

    Setting test=1 returns database query execution statistics

Flow.**packet_direction**()
    Return the direction of the current packet in the flow where c2s is client to server and s2c is server to client.

Flow.**packet_directions**(*test=0*)
    Return the set of packet directions of all packets recorded in the current flow, deduplicated for multiple switches. Returns a list of directions per packet where 1 = forward and 0 = reverse direction and oldest is the left most position and newest on the right

Flow.**packet_sizes**(*test=0*)
    Return the set of packet sizes of all packets recorded in the current flow, deduplicated for multiple switches. Returns a list of sizes per packet where the oldest is the left most position and newest on the right

Flow.**client**()
    Returns the IP that is the originator of the flow (if known, otherwise 0)

    Finds first packet seen for the flow_hash within the time limit and returns a the source IP

Flow.**origin**()
    Returns the IP and DPID that is the originator of the flow (if known, otherwise 0)

    Finds first packet seen for the flow_hash within the time limit and returns a tuple of the source IP and the dpid

Flow.**server**()
    The IP that is the destination of the flow (if known, otherwise 0)

    Finds first packet seen for the hash within the time limit and returns the destination IP

Flow.**max_packet_size**()
    Return the size of the largest packet in the flow (in either direction)

Flow.**max_interpacket_interval**()
    Return the size of the largest inter-packet time interval in the flow (assessed per direction in flow) as seconds (type float)

    Note: c2s = client to server direction s2c = server to client direction

    Note: results are slightly inaccurate due to floating point rounding.

Flow.**min_interpacket_interval**()
:   Return the size of the smallest inter-packet time interval in the flow (assessed per direction in flow) as seconds (type float)

    Note: c2s = client to server direction s2c = server to client direction

    Note: results are slightly inaccurate due to floating point rounding.

Flow.**not_suppressed**(*dpid*, *suppress_type*)
:   Check flow_mods to see if current flow context is already suppressed within suppression stand-down time for that switch, and if it is then return False, otherwise True

    The stand-down time is to reduce risk of overloading switch with duplicate suppression events.

    Called from nmeta.py

class Flow.**FlowMod**(*flow_mods*, *flow_hash*, *dpid*, *_type*, *standdown*)
:   Bases: object

    An object that represents an individual Flow Modification, used for recording the circumstances into the flow_mods MongoDB collection

    dbdict()
    :   Return a dictionary object of specific FlowMod parameters for storing in the database

    commit()
    :   Record removed mod into MongoDB flow_mods collection.

Flow.**record_suppression**(*dpid*, *suppress_type*, *result*, *standdown=0*)
:   Record that the flow is being suppressed on a particular switch in the flow_mods database collection, so that information is available to API consumers, such as the WebUI

identities module

The identities module is part of the nmeta suite

It provides an abstraction for participants (identities), using a MongoDB database for storage and data retention maintenance.

Identities are identified via TBD....

There are methods (see class docstring) that provide harvesting of identity metadata and various retrieval searches

class identities.**Identities**(*config*, *policy*)
:   Bases: baseclass.BaseClass

    An object that represents identity metadata

    Main function used to harvest identity metadata: (assumes class instantiated as an object called 'ident')

    ident.harvest(pkt, flow.packet) Passed a raw packet and packet metadata from flow object. Check a packet_in event and harvest any relevant identity indicators to metadata

    Functions available for Classifiers: (assumes class instantiated as an object called 'ident')

    ident.findbymac(mac_address) Look up identity object for a MAC address

    ident.findbynode(host_name) Look up identity object by host name (aka node) Additionally, can set:

    > regex=True Treat service_name as a regular expression harvest_type= Specify what type of harvest (i.e. DHCP)

ident.findbyservice(service_name) Look up identity object by service name Additionally, can set:

> regex=True Treat service_name as a regular expression harvest_type= Specify what type of harvest (i.e. DNS_A) ip_address= Look for specific IP address

See function docstrings for more information

class Identity
> Bases: `object`

An object that represents an individual Identity Indicator

dbdict()
> Return a dictionary object of identity metadata parameters for storing in the database

class Identities.**DHCPMessage**
> Bases: `object`

An object that represents an individual DHCP message. Used for storing DHCP state by recording DHCP events

dbdict()
> Return a dictionary object of dhcp message parameters for storing in the database

Identities.**harvest**(*pkt*, *flow_pkt*)
> Passed a raw packet and packet metadata from flow object. Check a packet_in event and harvest any relevant identity indicators to metadata

Identities.**harvest_arp**(*pkt*, *flow_pkt*)
> Harvest ARP identity metadata into database. Passed packet-in metadata from flow object. Check ARP reply and harvest identity indicators to metadata

Identities.**harvest_dhcp**(*flow_pkt*)
> Harvest DHCP identity metadata into database. Passed packet-in metadata from flow object. Check LLDP TLV fields and harvest any relevant identity indicators to metadata

Identities.**harvest_lldp**(*flow_pkt*)
> Harvest LLDP identity metadata into database. Passed packet-in metadata from flow object. Check LLDP TLV fields and harvest any relevant identity indicators to metadata

Identities.**harvest_dns**(*flow_pkt*)
> Harvest DNS identity metadata into database. Passed packet-in metadata from flow object. Check DNS answer(s) and harvest any relevant identity indicators to metadata

Identities.**findbymac**(*mac_addr*, *test=0*)
> Passed a MAC address and reverse search identities collection returning first match as a dictionary version of an Identity class, or empty dictionary if not found

> Setting test=1 returns database query execution statistics

Identities.**findbynode**(*host_name*, *harvest_type='any'*, *regex=False*, *test=0*)
> Find by node name Pass it the name of the node to search for. Additionally, can set:

> > regex=True Treat service_name as a regular expression harvest_type= Specify what type of harvest (i.e. DHCP)

> Returns a dictionary version of an Identity class, or 0 if not found

> Setting test=1 returns database query execution statistics

Identities.**findbyservice**(*service_name*, *harvest_type='any'*, *regex=False*, *ip_address='any'*, *test=0*)
> Find by service name Pass it the name of the service to search for. Additionally, can set:

regex=True Treat service_name as a regular expression harvest_type= Specify what type of harvest (i.e. DNS_A) ip_address= Look for specific IP address

Returns boolean

Setting test=1 returns database query execution statistics

identities.**mac_addr**(*address*)
    Convert a MAC address to a readable/printable string

forwarding module

This module is part of the nmeta suite running on top of Ryu SDN controller to provide network identity and flow metadata. It provides methods for forwarding functions.

class forwarding.**Forwarding**(*config*)
    Bases: baseclass.BaseClass

    This class is instantiated by nmeta.py and provides methods for making forwarding decisions and transformations to packets.

    basic_switch(*event*, *in_port*)
        Passed a packet in event and return an output port

switches module

This module is part of the nmeta suite running on top of Ryu SDN controller.

It provides classes that abstract the details of OpenFlow switches

class switches.**Switches**(*config*)
    Bases: baseclass.BaseClass

    This class provides an abstraction for a set of OpenFlow Switches.

    It stores instances of the Switch class in a dictionary keyed by DPID. The switch instances are accessible externally.

    A standard (not capped) MongoDB database collection is used to record switch details so that they can be accessed via the external API.

    add(*datapath*)
        Add a switch to the Switches class

    stats_reply(*msg*)
        Read in a switch stats reply

    delete(*datapath*)
        Delete a switch from the Switches class

class switches.**Switch**(*config*, *datapath*, *offset*)
    Bases: baseclass.BaseClass

    This class provides an abstraction for an OpenFlow Switch

    dbdict()
        Return a dictionary object of switch parameters for storing in the database

    request_switch_desc()
        Send an OpenFlow request to the switch asking it to send us it's description data

set_switch_config(*config_flags*, *miss_send_len*)
> Set config on a switch including config flags that instruct fragment handling behaviour and miss_send_len which controls the number of bytes sent to the controller when the output port is specified as the controller.

packet_out(*data*, *in_port*, *out_port*, *out_queue*, *no_queue=0*)
> Sends a supplied packet out switch port(s) in specific queue.

> Set no_queue=1 if want no queueing specified (i.e. for a flooded packet). Also use for Zodiac FX compatibility.

> Does not use Buffer IDs as they are unreliable resource.

set_switch_table_miss(*miss_send_len*)
> Set a table miss rule on table 0 to send packets to the controller. This is required for OF versions higher than v1.0

class switches.**FlowTables**(*config*, *datapath*, *offset*)
> Bases: baseclass.BaseClass

> This class provides an abstraction for the flow tables on an OpenFlow Switch

suppress_flow(*msg*, *in_port*, *out_port*, *out_queue*)
> Add flow entries to a switch to suppress further packet-in events while the flow is active.

> Prefer to do fine-grained match where possible. Install reverse matches as well for TCP flows.

> Do not install suppression for these types of flow: - DNS (want to harvest identity) - ARP (want to harvest identity) - DHCP (want to harvest identity) - LLDP (want to harvest identity)

drop_flow(*msg*)
> Add flow entry to a switch to suppress further packet-in events for a particular flow.

> Prefer to do fine-grained match where possible.

> TCP or UDP source ports are not matched as ephemeral

add_flow(*match_d*, *actions*, *priority*, *idle_timeout*, *hard_timeout*, *cookie*)
> Add a flow entry to a switch

actions(*out_port*, *out_queue*, *no_queue=0*)
> Create actions for a switch flow entry. Specify the out port and QoS queue, and set no_queue=1 if don't want QoS set. Returns a list of action objects

match_ipv4_tcp(*ipv4_src*, *ipv4_dst*, *tcp_src*, *tcp_dst*)
> Match an IPv4 TCP flow on a switch. Passed IPv4 and TCP parameters and return an OpenFlow match object for this flow

match_ipv4_udp(*ipv4_src*, *ipv4_dst*, *udp_src*, *udp_dst*)
> Match an IPv4 UDP flow on a switch. Passed IPv4 and UDP parameters and return an OpenFlow match object for this flow

match_ipv6_tcp(*ipv6_src*, *ipv6_dst*, *tcp_src*, *tcp_dst*)
> Match an IPv6 TCP flow on a switch. Passed IPv6 and TCP parameters and return an OpenFlow match object for this flow

match_ipv6_udp(*ipv6_src*, *ipv6_dst*, *udp_src*, *udp_dst*)
> Match an IPv6 UDP flow on a switch. Passed IPv6 and UDP parameters and return an OpenFlow match object for this flow

match_ipv4(*ipv4_src*, *ipv4_dst*, *ip_proto*)
> Match an IPv4 flow on a switch. Passed IPv4 parameters and return an OpenFlow match object for this flow

match_ipv6(*ipv6_src*, *ipv6_dst*)
    Match an IPv6 flow on a switch. Passed IPv6 parameters and return an OpenFlow match object for this flow

switches.**_ipv4_t2i**(*ip_text*)
    Turns an IPv4 address in text format into an integer. Borrowed from rest_router.py code

nethash module

The nethash module is part of the nmeta suite

It provides functions for hashing packets and flows to unique identifiers

nethash.**hash_flow**(*flow_5_tuple*)
    Generate a predictable flow_hash for the 5-tuple. For TCP the hash is the same no matter which direction the traffic is travelling for all packets that are part of that flow.

    Pass this function a 5-tuple.

    For TCP, this tuple should be: (ip_src, ip_dst, tp_src, tp_dst, ip_proto)

    For other IP packets, the tuple should be: (eth_src, eth_dst, dpid, packet_timestamp, ip_proto)

    For non-IP packets, the tuple should be: (eth_src, eth_dst, dpid, packet_timestamp, 0)

nethash.**hash_packet**(*packet*)
    Generate a hash of flows packet object for use in deduplication where the same packet is received from multiple switches.

    Retransmissions of a packet that is part of a flow should have same hash value, so that retransmissions can be measured.

    The packet hash is a unique unidirectional packet identifier

    For TCP packets, the hash is derived from: ip_src, ip_dst, proto, tp_src, tp_dst, tp_seq_src, tp_seq_dst

    For non-flow packets, the hash is derived from: eth_src, eth_dst, eth_type, dpid, timestamp

nethash.**hash_tuple**(*hash_tuple*)
    Simple function to hash a tuple with MD5. Returns a hash value for the tuple

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

Symbols

A

B

C

D

E

F

G