nmeta2 Documentation

Release 0.3.5

Matthew John Hayes

September 29, 2016

Contents

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

Contents:

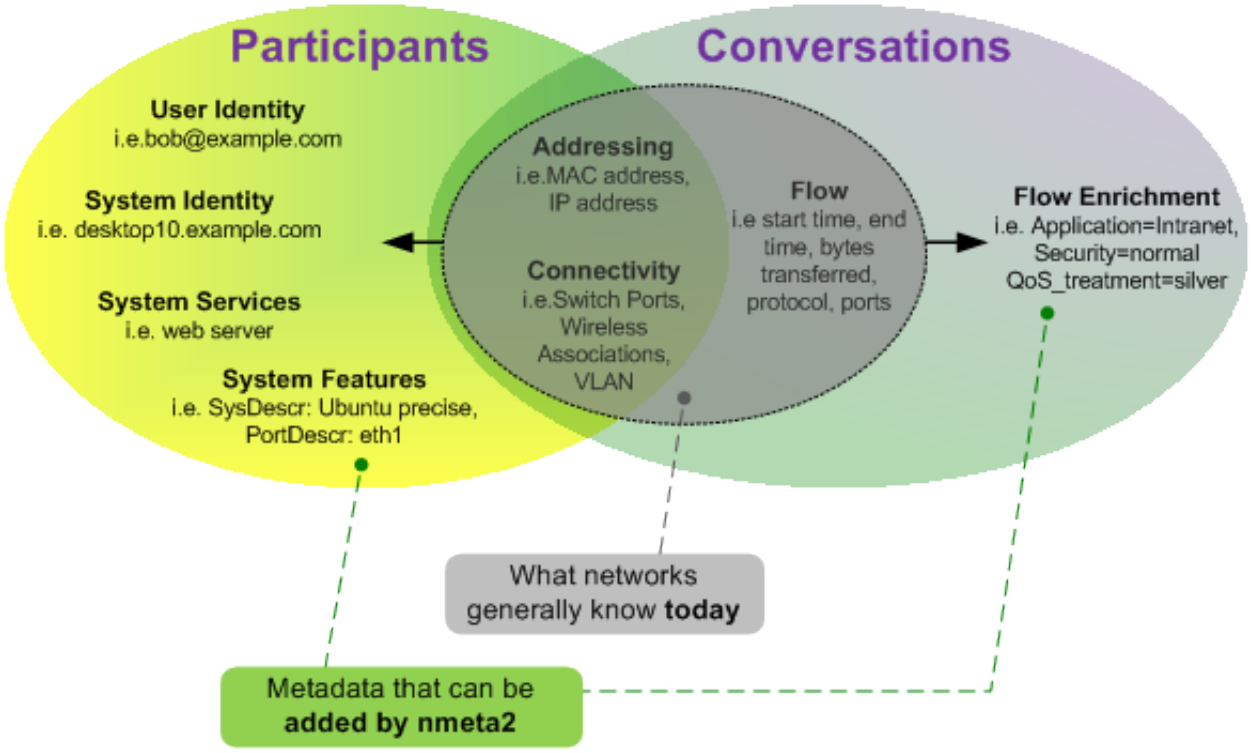
CHAPTER 1

Introduction

The nmeta2 *(pronounced en-meta two)* 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.

1.1 Background

Today, networks generally have only a limited view of participants and conversation types



The goal of the nmeta2 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.

Nmeta2 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/).

1.2 Distributed System

Nmeta2 distributes the heavy-lifting work of traffic classification to auxiliary devices, called a Data Plane Auxiliary Engines (DPAE), that scale horizontally.

See separate repository for DPAE

1.3 Limitations

Nmeta2 code is under construction, so a number of features are not implemented yet, or not finished.

1.4 Feature Enhancement Wishlist

See Issues for list of enhancements and bugs

1.5 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.

1.6 Disclaimer

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

1.7 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

CHAPTER 2

Install

This guide is for installing nmeta2 on Ubuntu OS.

2.1 Pre-Work

2.1.1 Ensure packages are up-to-date

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

2.1.2 Install Python pip

```
sudo apt-get install python-pip
```

2.1.3 Install git

Install git and git-flow for software version control:

```
sudo apt-get install git git-flow
```

2.2 Install Ryu OpenFlow Controller

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

```
sudo pip install ryu
```

2.3 Install Packages Required by nmeta

2.3.1 Install pytest

Pytest is used to run unit tests:

```
sudo apt-get install python-pytest
```

2.3.2 Install YAML

Install Python YAML ("YAML Ain't Markup Language") for parsing config and policy files:

```
sudo apt-get install python-yaml
```

2.3.3 Install simplejson

```
sudo pip install simplejson
```

2.3.4 Install mock

```
sudo pip install -U mock
```

2.4 Install MongoDB

MongoDB is the database used by nmeta2. Install MongoDB as per their instructions :

Import the MongoDB public GPG Key:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
```

Create a list file for MongoDB:

```
echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/s
```

Reload local package database:

```
sudo apt-get update
```

Install MongoDB:

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

Add pymongo for a Python API into MongoDB:

```
sudo apt-get install build-essential python-dev
sudo pip install pymongo
```

Turn on smallfiles to cope with small file system size:

```
sudo vi /etc/mongod.conf
```

Add this to the storage section of the config:

```
mmapv1:
  smallFiles: true
```

Start MongoDB (if required) with:

```
sudo service mongod start
```

2.5 Install nmeta2

Clone nmeta2

```
cd
git clone https://github.com/mattjhayes/nmeta2.git
```

2.6 Run nmeta2

```
cd
cd ryu
PYTHONPATH=. ./bin/ryu-manager ../nmeta2/nmeta2/nmeta2.py
```

2.7 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:

```
# Run nmeta:
alias nm2="cd; cd ryu; PYTHONPATH=. ./bin/ryu-manager ../nmeta2/nmeta2/nmeta2.py --log-config-file ~/
#
# Test nmeta:
alias nm2t="cd ~/nmeta2/test/; py.test"
```

CHAPTER 3

Code Documentation

3.1 api 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 RESTful API connectivity.

class api.**Api**(*_nmeta*, *_config*, *_wsgi*)
Bases: object

This class is instantiated by nmeta.py and provides methods for RESTful API connectivity.

IP_PATTERN = '\\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\\.|$){4}\\b'

url_dpae_base = '/nmeta/v2/aux/'

url_dpae_uuid = '/nmeta/v2/aux/{uri_uuid}'

url_dpae_uuid_keepalive = '/nmeta/v2/aux/{uri_uuid}/keepalive/'

url_dpae_uuid_main_policy = '/nmeta/v2/aux/{uri_uuid}/main_policy/'

url_dpae_uuid_sendconfpkt = '/nmeta/v2/aux/{uri_uuid}/send_conf_packet/'

url_dpae_uuid_tc_classify = '/nmeta/v2/aux/{uri_uuid}/services/tc/classify/'

url_dpae_uuid_tc_opt_rules = '/nmeta/v2/aux/{uri_uuid}/services/tc/opt_rules/'

url_dpae_uuid_tc_state = '/nmeta/v2/aux/{uri_uuid}/services/tc/state/'

url_idmac = '/nmeta/v2/id/mac/'

class api.**JSON_Body**(*req_body*)
Bases: object

Represents a JSON-encoded body of an HTTP request. Doesn't do logging, but does set .error when things don't go to plan with a friendly message.

decode(*req_body*)
Passed an allegedly JSON body and see if it decodes. Set error variable for exceptions

validate(*key_list*)
Passed a list of keys and check that they exist in the JSON. If they don't return 0 and set error to description of first missing key that was found

exception api.**NotFoundError**(*msg=None*, ***kwargs*)
Bases: ryu.exception.RyuException

message = 'Error occurred talking to function <TBD>'

class api.**RESTAPIController**(*req*, *link*, *data*, ***config*)
Bases: ryu.app.wsgi.ControllerBase

This class is used to control REST API access to the nmeta data and control functions

rest_dpae_create(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_delete(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_keepalive(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_main_policy_read(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_read(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_read_uuid(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_send_sniff_conf_pkt(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_tc_classify_advice(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_tc_opt_rules_read(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_dpae_tc_state_update(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

rest_idmac_read(**args*, ***kwargs*)
Run a REST command and return appropriate response. Keys/Values returned to this wrapper in a dictionary. Valid Keys are:

'msg': the data to return in the message body 'location': a new location for the resource 'status': HTTP status code to return

api.**rest_command**(*func*)
REST API command template

api.**version_compare**(*version1*, *version2*)
Compare two semantic version numbers and return 1 if they are the same major version number

3.2 config module

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

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

class config.**Config**
Bases: object

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'

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

set_value(*config_key*, *config_value*)
Passed a key and see if it exists in the config YAML. If it does then set its value to the supplied value and return 1 otherwise 0

3.3 main_policy 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 an object for the main policy and includes ingesting the policy from file on class instatiation and validating its syntax.

class main_policy.**Identity**(*logger*, *policy*)
Bases: object

Represents the portion of main policy off the root key 'identity'

IDENTITY_KEYS = ('arp', 'lldp', 'dns', 'dhcp')

class main_policy.**MainPolicy**(*_config*)
Bases: object

This class is instantiated by nmeta2.py and provides methods to ingest the policy file main_policy.yaml and validate that it is correctly structured . Directly accessible values to read:

main_policy # main policy YAML object tc_policies.mode # mode for DPAE connectivity (active or passive) identity.arp # True if identity arp harvest is enabled identity.lldp # True if identity lldp harvest is enabled identity.dns # True if identity dns harvest is enabled identity.dhcp # True if identity dhcp harvest is enabled

Methods: <TBD> tc_policies.* tc_rules.* identity.* qos_treatment.get_policy_qos_treatment_value(key) port_sets.get_tc_ports(dpid) # Get ports for a DPID to run TC on optimised_rules.get_rules() # Get optimised TC rules to install

Public Functions: validate_keys(logger, keys, schema, branch) validate_value(logger, key, value, schema, branch) is_valid_macaddress(logger, value_to_check) is_valid_ethertype(logger, value_to_check) is_valid_ip_space(logger, value_to_check) is_valid_transport_port(logger, value_to_check)

TOP_KEYS = ('tc_policies', 'tc_rules', 'identity', 'qos_treatment', 'port_sets')

ingest_policy(*config_directory*, *main_policy_filename*)
Read in main policy from file

class main_policy.**Optimise**(*logger*, *policy*)
Bases: object

Represents an optimised set of TC rules to install on a switch

CONDITION_TYPES = {'identity_service_dns_re': 'identity', 'ip_dst': 'static', 'identity_service_dns': 'identity', 'tcp_src'

get_rules()
Return an optimised flow entry match set to install to switches based on the tc_rules

class main_policy.**PortSets**(*logger*, *policy*)
Bases: object

Represents the portion of main policy off the root key 'port_sets'

get_tc_ports(*dpid*)
Passed a DPID and return a tuple of port numbers on which to run TC on that switch, or 0 if none

class main_policy.**QoSTreatment**(*logger*, *policy*)
Bases: object

Represents the portion of main policy off the root key 'qos_treatment'

QOS_TREATMENT_KEYS = ('default_priority', 'constrained_bw', 'high_priority', 'low_priority')

get_policy_qos_treatment_value(*qos_key*)
Return a value for a given key under the 'qos_treatment' root of the policy

class main_policy.**TCPolicies**(*logger*, *policy*)
Bases: object

Represents the portion of main policy off the root key 'tc_policies'

TC_POLICY_KEYS = ('comment', 'rule_set', 'port_set', 'mode')

TC_POLICY_MODE_VALUES = ('active', 'passive')

class main_policy.**TCRule**(*logger*, *rule*)
Bases: object

Represents a TC rule

TC_CONFIG_CONDITIONS = {'match_type': 'MatchType', 'identity_service_dns_re': 'String', 'ip_dst': 'IPAddressSpac

TC_CONFIG_MATCH_TYPES = ('any', 'all')

TC_RULE_ATTRIBUTES = ('comment', 'match_type', 'conditions_list', 'actions')

class main_policy.**TCRules**(*logger*, *policy*)
Bases: object

Represents the portion of main policy off the root key 'tc_rules'

main_policy.**is_valid_ethertype**(*logger*, *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

main_policy.**is_valid_ip_space**(*logger*, *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

main_policy.**is_valid_macaddress**(*logger*, *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

main_policy.**is_valid_transport_port**(*logger*, *value_to_check*)
Passed a logger ref and 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

main_policy.**validate_keys**(*logger*, *keys*, *schema*, *branch*)
Validate a set of keys against a schema tuple to ensure that there are no missing or extraneous keys

main_policy.**validate_value**(*logger*, *key*, *value*, *schema*, *branch*)
validate that the value complies with the schema

3.4 nmeta2 module

This is the main module of the nmeta2 suite running on top of the Ryu SDN controller to provide network identity and flow (traffic classification) metadata. . It supports OpenFlow v1.3 switches and Data Path Auxiliary Engines (DPAE) . Do not use this code for production deployments - it is proof of concept code and carries no warrantee whatsoever. . You have been warned.

class nmeta2.**Nmeta**(**args*, ***kwargs*)
Bases: ryu.base.app_manager.RyuApp

This is the main class of nmeta2 and is run by Ryu

OFP_VERSIONS = [4]

desc_stats_reply_handler(*ev*)
Receive a reply from a switch to a description statistics request

dpae_join(*pkt*, *datapath*, *in_port*)
A DPAE may have sent us a join discovery packet (Phase 2) Check the packet payload to see if it is valid

error_msg_handler(*ev*)
A switch has sent us an error event

flow_removed_handler(*ev*)
A switch has sent an event to us because it has removed a flow from a flow table

switch_connection_handler(*ev*)
A switch has connected to the SDN controller. We need to do some tasks to set the switch up properly:

•Instantiate a class to represent the switch and flow tables

•Delete all existing flow entries

•Set config for fragment handling and table miss packet length

•Set up initial flow entries in flow tables

•Install non-DPAE TC flows from optimised policy to switch

•Request the switch send us its description

Supported OpenFlow versions is controlled by the OFP_VERSIONS constant set in class base.

tc_advice_id(*dpid*, *tc_type*, *tc_subtype*, *src_mac*, *detail1*)
Process a Traffic Classification advice message from a DPAE that relates to an identity

tc_start(*datapath*, *dpae_port*)
Add a Flow Entry to switch to clone selected packets to a DPAE so that it can perform Traffic Classification analysis on them

3.5 of_error_decode module

This module is part of the nmeta suite running on top of Ryu SDN controller. It decodes OpenFlow error messages . Example usage:

import of_error_decode ... type1, type2, code1, code2 = of_error_decode.decode(error_type,

error_code)

self.logger.error('error_type=%s %s error_code=%s %s', type1, type2, code1, code2)

of_error_decode.**decode**(*error_type*, *error_code*)
Return a decoded explaination of an OpenFlow error type/code

3.6 switch_abstraction module

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

It provides functions that abstract the details of OpenFlow switches

class switch_abstraction.**FlowTables**(*_nmeta*, *logger*, *_config*, *datapath*)
Bases: object

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

add_fe_amf_macport_dst(*dpae_port*, *eth_dst*)
Add Flow Entry (FE) to the Active Mode Filter flow table to send packets destined for learned destination MACs out the port to the DPAE

add_fe_amf_miss()
Add Active Mode Filter flow table miss Flow Entry to send packets to next flow table

add_fe_fwd_macport_dst(*out_port*, *eth_dst*)
Add Flow Entry (FE) to the Forwarding flow table to match learned destination MAC and output learned port to avoid flooding

add_fe_fwd_miss()
Add Forwarding flow table miss Flow Entry to flood packets out ports as we haven't learnt the MAC address

add_fe_iig_broadcast()
Add Flow Entry (FE) to the Identity Indicators (General) flow table to flood Ethernet broadcast packets to lower the load on the rest of the pipeline

add_fe_iig_dhcp(*dpae_port*)
Add Flow Entry (FE) to the Identity Indicators (General) flow table to clone DHCP packets to a DPAE

add_fe_iig_dns(*dpae_port*)
Add Flow Entry (FE) to the Identity Indicators (General) flow table to clone DNS packets to a DPAE

add_fe_iig_lldp(*dpae_port*)
Add Flow Entry (FE) to the Identity Indicators (General) flow table to clone LLDP packets to a DPAE

add_fe_iig_miss()
Add Identity Indicator (General) flow table miss Flow Entry to continue pipeline processing

add_fe_iim_dpae_active_bypass(*dpae_port*)
Add Identity Indicator (MAC) Flow Entry to bypass intermediate tables for traffic from DPAE (return packets from active mode TC) and goto treatment table direct

add_fe_iim_dpae_join()
Add Identity Indicator (MAC) Flow Entry to send DPAE Join packets to the controller as packet-in messages

add_fe_iim_macport_src(*in_port*, *eth_src*)
Add Flow Entry (FE) to the Identity Indicator (MAC) flow table to match combo of in port and source MAC and goto next table. This is used filter punts to the controller for learning MAC to port mappings so that only new port/MAC mappings that aren't matched by a rule are punted by the Identity Indicator (MAC) flow table miss rule

add_fe_iim_miss()
Add Identity Indicator (MAC) flow table miss Flow Entry to clone a table-miss packet to the controller as a packet-in message

add_fe_tc_dpae(*tc_flows*, *dpae_port*, *mode*)
Install DPAE Traffic Classification (TC) flows from optimised TC policy to switch (i.e. Flow Entries that invoke actions that need for DPAE to classify). Mode is either active or passive. Passive Mode: Output to DPAE and Goto-Table Traffic Treatment (ft_tt) Active Mode: Goto-Table Active Mode Filter (ft_amf)

add_fe_tc_id(*id_type*, *id_detail*, *id_mac*, *tc_flows*)
Add Flow Entri(es) to the switch if required for identity match and action. Check to see if we have any to install, and if so use a separate function to install to switch

add_fe_tc_id_install(*id_mac*, *action*)
Add Flow Entry to the switch for identity match and action

add_fe_tc_miss()
Add Traffic Classification flow table miss Flow Entry to send packets to Traffic Treatment Flow Table

add_fe_tc_static(*tc_flows*)
Install non-DPAE static Traffic Classification (TC) flows from optimised TC policy to switch (i.e. Flow Entries that invoke actions without need for DPAE to classify)

add_fe_tcf_accepts()
Add Traffic Classification Filter flow table accept Flow Entries to send packets to TC flow table)

add_fe_tcf_miss()
Add Traffic Classification Filter flow table miss Flow Entry to send packets to Traffic Treatment (ft_tt) table

add_fe_tcf_suppress(*suppress_dict*)
Process a Traffic Classification flow suppression request from a DPAE, where it has requested that we don't send any more packets to it for a specific flow. Install an FE to switch for each direction of the flow to bypass sending to DPAE. . Only supports IPv4 and TCP at this stage. .

add_fe_tt_advised(*flow_dict*)
Process a Traffic Classification flow treatment advice from a DPAE. Install an FE to switch for each direction of the flow applying the appropriate treatment. . Only supports IPv4 and TCP at this stage. .

add_fe_tt_miss()
Add Traffic Treatment flow table miss Flow Entry to send packets to next flow table

add_group_dpae(*dpae_port*)
Add Group Table to the switch for forwarding packets to DPAE out a specific port. Note, will generate error if group table already exists.

delete_all_flows()
Delete all Flow Entries from all Flow Tables on this switch

delete_fe(*match*, *flow_table_id*)
Delete a specific FE from a specific Flow Table on this switch

matches_add_fe_prereqs(*fe_matches*)
Passed a dictionary of match_type, value pairs for creating a flow entry on a switch and work out what match types are missing as per requirements of OpenFlow v1.3 standard and add them to the dictionary

class switch_abstraction.**MACTable**(*_nmeta*, *logger*, *_config*, *datapath*)
Bases: object

This class provides an abstraction for the MAC table on an OpenFlow Switch that maps MAC addresses to switch ports

add(*mac*, *in_port*, *context*)
Passed a MAC address and the switch port it was learnt via along with a context. Add this to the database and tidy up by removing any other entries for this MAC on this switch in given context.

delete(*mac*, *in_port*, *context*)
Passed a MAC address, switch port and context to delete. Delete any entries from DB and from switch

dump_macs(*context*)
Return a list of all known MAC addresses for a given context on this switch

mac2port(*mac*, *context*)
Passed a dpid and mac address and return the switch port number that this mac has been learned via (or 999999999 if unknown)

class switch_abstraction.**Switch**(*_nmeta*, *logger*, *_config*, *datapath*)
Bases: object

This class provides an abstraction for an OpenFlow Switch

get_friendly_of_version(*ofproto*)
Passed an OF Protocol object and return a human-friendly version of the protocol revision number

packet_out(*data*, *in_port*, *out_port*, *out_queue*, *nq=0*)
Sends a supplied packet out switch port(s) in specific queue. Set nq=1 if want no queueing specified (i.e. for a flooded packet) Use nq=1 for Zodiac FX compatibility Does not support use of Buffer IDs

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.

class switch_abstraction.**Switches**(*_nmeta*, *_config*)
Bases: object

This class provides an abstraction for a set of OpenFlow Switches

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

datapath(*dpid*)
Return a datapath for a given switch dpid

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

a

c

m

n

o

s

A

C

D

E

F

G

I

J

M

N

O

P

Q

R

S

T

U

V