Static Entry Pusher API

This guide is for Floodlight v1.0 and up. If you are using Floodlight v0.91 or earlier or are using a master copy downloaded prior to December 30, 2014, please consider upgrading to the latest release. Please refer to these instructions for help using the Static Flow Pusher for pre-v1.0 releases.

Floodlight master branch as of mid-June 2016 supports pushing groups in addition to flows! We encourage you to upgrade if you're interested in this feature.

Support for OpenFlow 1.5 matches, actions, and instructions has also been added.

Table of Contents

- 1 Naming of this Module
- 2 What is the Static Entry Pusher?
  - 2.1 Proactive vs Reactive Entry Insertion
- 3 How is the Static Entry Pusher Used?
  - 3.1 API Summary
  - 3.2 Adding an Entry
    - 3.2.1 Adding a Flow
    - 3.2.2 Adding a Group
  - 3.3 Listing Entries
  - 3.4 Clearing Entries
  - 3.5 Deleting Entries
- 4 Entry Composition
  - 4.1 Common to Both Flow and Group Entries
    - 4.1.1 Required Entry Properties
    - 4.1.2 Optional Entry Properties
    - 4.1.3 Reserved Port Keywords
    - 4.1.4 Reserved Group Keywords
    - 4.1.5 Optional Actions
  - 4.2 Flow Entries
    - 4.2.1 Optional Flow Entry Properties
    - 4.2.2 Optional Flow Entry Match Fields
    - 4.2.3 Optional Action and Instruction Fields
    - 4.2.4 Copy Field Action
      - 4.2.4.1 Example Copy Field Action
    - 4.2.5 Stat Trigger Instruction
      - 4.2.5.1 Thresholds
      - 4.2.5.2 Example Stat Trigger Instruction
  - 4.3 Group Entries
    - 4.3.1 Required Group Entry Properties
    - 4.3.2 Required Group Bucket Properties

        - 4.3.3 Example Group Entry
- 5 Using Static Entry Pusher in Practice
    - 5.1 Can Use Python to Push Entries
    - 5.2 Using the OpenFlow 1.2+ set_field Action

Naming of this Module

Since its inception, the Floodlight controller has provided the ability to install flow table entries via the REST API. This feature was originally debuted as the Static Flow Entry Pusher. However, many if not most developers and users simply referred to this module as the Static Flow Pusher. Now that the Static Flow Pusher also supports the installation of group table entries, the term "flow" in the name is misleading. Thus, going forward, the module will be referred to as the Static Entry Pusher. (Note that the term "static" is also misleading, as the module supports entries with timeouts, but this "non-static" feature is seldom used.)

What is the Static Entry Pusher?

The Static Entry Pusher is a Floodlight module, exposed via a REST API, that allows a user to manually insert flows and groups into an OpenFlow network.

Proactive vs Reactive Entry Insertion

OpenFlow supports two methods of entry insertion: proactive and reactive. Reactive entry insertion occurs when a packet reaches an OpenFlow switch without a matching flow. The packet is sent to the controller, which evaluates it, adds the appropriate entries, and lets the switch continue its forwarding. Alternatively, entries can be inserted proactively by the controller in switches before packets arrive. In the case of proactive flow insertion, the arriving packet will never be sent to the controller for evaluation since it matches the proactively-inserted flow.

Floodlight supports both mechanisms of entry insertion. The Static Entry Pusher is generally useful for proactive entry insertion.

Note that by default, Floodlight loads the Forwarding module which does reactive entry pushing. If you would like to exclusively use static entries, you must remove Forwarding from the floodlight.properties file.

How is the Static Entry Pusher Used?

API Summary

Multiple URIs are available in Floodlight master June 2016 and later. This is for backwards compatibility with folks who have northbound applications that use the URIs present in past controller versions.

Description	URI	Arguments
Add/Delete entry	/wm/<module>/json	<module> see table below
List entries	/wm/<module>/list/<switch>/json	<module> see table below <switch> is a valid switch DPID formatted as: -- hex-string, e.g. "00:00:00:00:00:00:00:0c" -- integer, e.g. "12" -- keyword "all" for all switches
Clear entries	/wm/<module>/clear/<switch>/json	<module> see table below <switch> is a valid switch DPID formatted as: -- hex-string, e.g. "00:00:00:00:00:00:00:0c" -- integer, e.g. "12" -- keyword "all" for all switches

The evolution of the Static Entry Pusher module has resulted in modification of the URI over time. Hopefully "staticentrypusher" will stick going forward. Use the following in place of <module> based on your Floodlight version:

<module> in URI	Supported Controller Version(s)
staticentrypusher	master
staticflowpusher	v1.0 - v1.2; master (deprecated)
staticflowentrypusher	v0.91- (see old docs); master (deprecated)

Adding an Entry

The Static Entry Pusher is accessible via a REST API. To add a static entry, the user needs to define the entry in JSON format.

Adding a Flow

For example, to insert a flow on switch 1 that takes packets from port 1 and outputs them on port 2, you can compose the JSON string and simply use a curl command to send the HTTP POST to the controller. The second command will dump the flow so you can see it set.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "name":"flow-mod-1",
"cookie":"0", "priority":"32768", "in_port":"1","active":"true",
"actions":"output=2"}' http://<controller_ip>:8080/wm/staticentrypusher/json
```

Adding a Group

To add a group, the process is very similar to adding a flow. Note the required "entry_type" field to tell the Static Entry Pusher that the data in your message represents a group, not a flow. (If unspecified, "entry_type" defaults to "flow".) Although OpenFlow does not define bucket IDs, a group entry must have a unique bucket ID per bucket to define ordering of the buckets. Buckets are ordered sequentially from lowest ID to highest ID.

```
curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "entry_type":"group",
"name":"group-mod-1", "active":"true", "group_type":"select", "group_id":"1",
"group_buckets":[ {"bucket_id":"1", "bucket_watch_group":"any",
"bucket_weight":"50", "bucket_actions":"output=2"}, {"bucket_id":"2",
"bucket_watch_group":"any", "bucket_weight":"50",
"bucket_actions":"output=3"} ]}'
http://<controller_ip>:8080/wm/staticentrypusher/json
```

Listing Entries

To get a list of all static entries, the Static Entry Pusher REST API accepts an HTTP GET to the path specified in API Summary above. You can query for static flows on a per-switch basis or ask the controller for all static entries across all switches.

```
curl
http://<controller_ip>:8080/wm/staticentrypusher/list/00:00:00:00:00:00:00:01
/json
curl http://<controller_ip>:8080/wm/staticentrypusher/list/all/json
```

Clearing Entries

To clear entries per switch or globally:

```
curl
http://<controller_ip>:8080/wm/staticentrypusher/clear/00:00:00:00:00:00:00:0
1/json
curl http://<controller_ip>:8080/wm/staticentrypusher/clear/all/json
```

Deleting Entries

To delete a static entry you send an HTTP DELETE that includes the name of the entry used during insertion.

```
curl -X DELETE -d '{"name":"flow-mod-1"}'
http://<controller_ip>:8080/wm/staticentrypusher/json
```

Entry Composition

The syntax to compose a flow versus a group entry varies due to the different structure of each. To avoid presenting information in duplicate, this section first presents required and optional properties for both flow and group entries. The sections following this dive into properties only defined for flows and only defined for groups.

Common to Both Flow and Group Entries

The following tables define properties for both flow and group entries.

Required Entry Properties

Key	Value	Notes
name	<string>	Name of the entry. This is used as the primary key for the entry. **The name must be globally unique.**
switch	<switch DPID>	DPID of the switch to which this entry should be added. xx:xx:xx:xx:xx:xx:xx:xx
entry_type	<string>	"flow" for flow entries (default; can omit key) "group" for group entries (required)

Optional Entry Properties

Key	Value	Notes
active	<boolean>	"true" or "false"

Reserved Port Keywords

<reserved_port>	Notes
all	All switch ports
controller	Controller(s)
flood	All ports except ingress port and those disabled for flooding, e.g. by STP
in_port	Packet's ingress port
local	Local network stack of switch; i.e. to/from switch OS
normal	Process using switch's normal L2/L3 pipeline
any	Any switch port

Reserved Group Keywords

<reserved_group>	Notes
all	All switch groups
any	Any switch group

Optional Actions

Key	Value	Prerequisite Matches	OpenFlow Versions	Notes
output	<number> or <reserved_port>		all	No "drop" option. (Instead, specify no action to drop packets.) Can be hexadecimal (with leading 0x) or decimal. See table above for <reserved_port> values.
group	<number>		OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
enqueue	<number>:<number>		OpenFlow 1.0	First number is port number, second is queue ID. Can be hexadecimal (with leading 0x) or decimal.
set_queue	<number>		OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.

strip_vlan		eth_vlan_vid = something	OpenFlow 1.0	
set_vlan_vid	<number>	eth_vlan_vid = something	OpenFlow 1.0	Can be hexadecimal (with leading 0x) or decimal.
set_vlan_pcp	<number>	eth_vlan_vid = something	all	Can be hexadecimal (with leading 0x) or decimal.
push_vlan	<eth-type-number>		OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal. Must be followed by a separate action to set the VLAN VID.
pop_vlan		eth_vlan_vid = something	OpenFlow 1.1+	
set_eth_src	<MAC address>		OpenFlow 1.0 - 1.1	xx:xx:xx:xx:xx:xx
set_eth_dst	<MAC address>		OpenFlow 1.0 - 1.1	xx:xx:xx:xx:xx:xx
set_ip_tos	<number>	eth_type = 0x0800 || 0x86dd	OpenFlow 1.0	Can be hexadecimal (with leading 0x) or decimal.
set_ip_ecn	<number>	eth_type = 0x0800 || 0x86dd	OpenFlow 1.1	Can be hexadecimal (with leading 0x) or decimal.
set_ipv4_src	<IPv4 address>	eth_type = 0x0800	OpenFlow 1.0 - 1.1	xx.xx.xx.xx
set_ipv4_dst	<IPv4 address>	eth_type = 0x0800	OpenFlow 1.0 - 1.1	xx.xx.xx.xx
set_ip_ttl	<number>	eth_type = 0x0800 || 0x86dd	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
dec_ip_ttl		eth_type = 0x0800 || 0x86dd	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
copy_ip_ttl_in		eth_type = 0x0800 || 0x86dd	OpenFlow 1.1+	

copy_ip_ttl_out		eth_type = 0x0800 || 0x86dd	OpenFlow 1.1+	
set_mpls_label	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
set_mpls_tc	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
set_mpls_ttl	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
dec_mpls_ttl		eth_type = 0x8847 || 0x8848	OpenFlow 1.1+	
push_mpls	<number>		OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
pop_mpls	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
push_pbb	<number>		OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
pop_pbb		eth_type = 0x88e7	OpenFlow 1.1+	
set_tp_src	<number>	ip_proto = 0x06 || 0x11 || 0x84	OpenFlow 1.0	Can be hexadecimal (with leading 0x) or decimal.
set_tp_dst	<number>	ip_proto = 0x06 || 0x11 || 0x84	OpenFlow 1.0	Can be hexadecimal (with leading 0x) or decimal.
set_field	<OXM->value>	See match table above.	OpenFlow 1.2+	e.g. "set_field=eth_src->00:11:22:33:44:55"
meter	<number>		OpenFlow 1.5+	Replaces "instruction_goto_meter" Can be hexadecimal (with leading 0x) or decimal.

copy_field	<copy field>		OpenFlow 1.5+	JSON object. See copy field table.

Flow Entries

The following tables define properties for flow entries only.

Optional Flow Entry Properties

Key	Value	Notes
priority	<number>	Default is 32767. Max is 32767.
table	<number>	Default flow table is used if omitted.
idle_timeout	<number>	Default of zero, which is no-timeout.
hard_timeout	<number>	Default of zero, which is no-timeout.
cookie	<number>	Can be hexadecimal (with leading 0x) or decimal.
cookie_mask	<number>	Can be hexadecimal (with leading 0x) or decimal.

Optional Flow Entry Match Fields

Key	Value	Prerequisite Matches	OpenFlow Versions	Notes
in_port	<number> <reserved_port>		all	Switch port on which the packet is received Can be hexadecimal (with leading 0x) or decimal. See reserved port table.
eth_type	<number>		all	Can be hexadecimal (with leading 0x) or decimal.
eth_src	<MAC address>		all	xx:xx:xx:xx:xx:xx
eth_dst	<MAC address>		all	xx:xx:xx:xx:xx:xx

eth_vlan_vid	<number>		all	Must include 'present' bit (bit 12) in the mat value. E.g. to match on tag 0x0064, use the match 0x1064. Can be hexadecimal (with leading 0x) or decimal.
eth_vlan_pcp	<number>	eth_vlan_vid = something	all	Can be hexadecimal (with leading 0x) or decimal.
ip_proto	<number>	eth_type = 0x0800 || 0x86dd	all	Can be hexadecimal (with leading 0x) or decimal.
ipv4_src	<IPv4 address[/mask]>	eth_type = 0x0800	all	xx.xx.xx.xx[/xx]
ipv4_dst	<IPv4 address[/mask]>	eth_type = 0x0800	all	xx.xx.xx.xx[/xx]
ipv6_src	<IPv6 address[/mask]>	eth_type = 0x86dd	OpenFlow 1.2+	xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxx
ipv6_dst	<IPv6 address[/mask]>	eth_type = 0x86dd	OpenFlow 1.2+	xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxx
ipv6_label	<number>	eth_type = 0x86dd	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
ip_tos	<number>	eth_type = 0x0800 || 0x86dd	all	Can be hexadecimal (with leading 0x) or decimal.
ip_ecn	<number>	eth_type = 0x0800 || 0x86dd	all	Can be hexadecimal (with leading 0x) or decimal.
ip_dscp	<number>	eth_type = 0x0800 || 0x86dd	all	Can be hexadecimal (with leading 0x) or decimal.
tp_src	<number>	ip_proto = 0x06 || 0x11 || 0x84	all	UDP, TCP, and SCTP supported.
tp_dst	<number>	ip_proto = 0x06 || 0x11 || 0x84	all	UDP, TCP, and SCTP supported.
udp_src	<number>	ip_proto = 0x11	all	UDP only.

udp_dst	<number>	ip_proto = 0x11	all	UDP only.
tcp_src	<number>	ip_proto = 0x06	all	TCP only.
tcp_dst	<number>	ip_proto = 0x06	all	TCP only.
sctp_src	<number>	ip_proto = 0x84	all	SCTP only.
sctp_dst	<number>	ip_proto = 0x84	all	SCTP only.
icmpv4_type	<number>	ip_proto = 0x01	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
icmpv4_code	<number>	ip_proto = 0x01	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
icmpv6_type	<number>	ip_proto = 0x3a	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
icmpv6_code	<number>	ip_proto = 0x3a	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
ipv6_nd_sll	<MAC address>	icmpv6_type = 0x87	OpenFlow 1.2+	xx:xx:xx:xx:xx:xx
ipv6_nd_tll	<MAC address>	icmpv6_type = 0x88	OpenFlow 1.2+	xx:xx:xx:xx:xx:xx
ipv6_nd_target	<IPv6 address>	icmpv6_type = 0x87 || 0x88	OpenFlow 1.2+	xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxx
ipv6_exthdr	<number>	eth_type = 0x86dd	OpenFlow 1.3+	Can be hexadecimal (with leading 0x) or decimal.
arp_opcode	<number>	eth_type = 0x0806	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
arp_sha	<MAC address>	eth_type = 0x0806	OpenFlow 1.2+	xx:xx:xx:xx:xx:xx
arp_tha	<MAC address>	eth_type = 0x0806	OpenFlow 1.2+	xx:xx:xx:xx:xx:xx
arp_spa	<IPv4 address>	eth_type = 0x0806	OpenFlow 1.2+	xx.xx.xx.xx

arp_tpa	<IPv4 address>	eth_type = 0x0806	OpenFlow 1.2+	xx.xx.xx.xx
mpls_label	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
mpls_tc	<number>	eth_type = 0x8847 || 0x8848	OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
mpls_bos	<boolean>	eth_type = 0x8847 || 0x8848	OpenFlow 1.3+	"true" or "false"
pbb_uca	<boolean>		OpenFlow 1.3+	"true" or "false"
tunnel_id	<number>		OpenFlow 1.3+	Can be hexadecimal (with leading 0x) or decimal.
tunnel_ipv4_src	<IPv4 address>		OpenFlow 1.3+	xx.xx.xx.xx NXM used by OVS.
tunnel_ipv4_dst	<IPv4 address>		OpenFlow 1.3+	xx.xx.xx.xx NXM used by OVS.
metadata	<number>		OpenFlow 1.2+	Can be hexadecimal (with leading 0x) or decimal.
tcp_flags	<number>	ip_proto = 0x06	OpenFlow 1.5+	Can be hexadecimal (with leading 0x) or decimal.
actset_output	<number> <reserved_port>		OpenFlow 1.5+	Can be hexadecimal (with leading 0x) or decimal.
packet_type	<number>/<number>		OpenFlow 1.5+	Must take form ns/ns_type, where both ns a ns_type are numbers. Can be hexadecimal (with leading 0x) or decimal.

Optional Action and Instruction Fields

Key	Value	OpenFlow Versions	Notes

actions	<list of actions> <action1=value1,action2=value2,...>	OpenFlow 1.0 (technically)	Can be used in place of instruction_apply_actions for OpenFlow 1.1+. See action table.
instruction_apply_actions	<list of actions> <action1=value1,action2=value2,...>	OpenFlow 1.1+	OpenFlow 1.2+ uses OXM (match) syntax to specify set_field actions. See action table.
instruction_write_actions	<list of actions> <action1=value1,action2=value2,...>	OpenFlow 1.1+	OpenFlow 1.2+ uses OXM (match) syntax to specify set_field actions. See action table.
instruction_clear_actions		OpenFlow 1.1+	
instruction_write_metadata	<number>	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
instruction_goto_table	<number>	OpenFlow 1.1+	Can be hexadecimal (with leading 0x) or decimal.
instruction_goto_meter	<number>	OpenFlow 1.3 - 1.4	Replaced by "meter" action in OpenFlow 1.5.
instruction_stat_trigger	<stat trigger>	OpenFlow 1.5+	JSON object. See stat trigger table.

Copy Field Action

The copy_field action value is represented as a JSON object with the following keys and values:

Key	Value	Notes
src_field	<string>	The field to copy bits from. Any match field string key.
dst_field	<string>	The field to copy bits to. Any match field string key.

src_offset_bits	<number>	Offset in bits from location of src_field. Can be hexadecimal (with leading 0x) or decimal.
dst_offset_bits	<number>	Offset in bits from location of dst_field. Can be hexadecimal (with leading 0x) or decimal.
num_bits	<number>	Number of bits to copy. Can be hexadecimal (with leading 0x) or decimal.

Example Copy Field Action

An example copy_field JSON object might look like:

```
{
        "src_field" : "eth_src",
        "dst_field" : "eth_dst",
        "src_offset_bits" : "0",
        "dst_offset_bits" : "0",
        "num_bits" : "48"
}
```

Stat Trigger Instruction

The instruction_stat_trigger instruction value is represented as a JSON object with the following keys and values:

Key	Value	Notes
flags	<array of strings>	Possible flags: • "periodic" • "only_first"
thresholds	<array of thresholds>	Threshold objects are defined in the table below.

Thresholds

The following defines a threshold object:

Key	Value	Notes

oxs_type	<string>	Possible OXS types: • "byte_count" • "duration" (seconds) • "flow_count" • "idle_time" (seconds) • "packet_count"
value	<number>	Can be hexadecimal (with leading 0x) or decimal.

Example Stat Trigger Instruction

An example instruction_stat_type value might look like this:

```
{
        "flags" : [
                "periodic"
        ],
        "thresholds" : [
                {
                        "oxs_type" : "byte_count",
                        "value" : "1000"
                },
                {
                        "oxs_type" : "duration",
                        "value" : "60"
                }
        ]
}
```

Group Entries

The following tables define properties for group entries only.

Required Group Entry Properties

Key	Value	Notes
group_id	<number>	Can be hexadecimal (with leading 0x) or decimal.
group_type	<string>	"all", "select", "indirect", or "fast_failover".
group_buckets	<list of buckets> <[{bucket1}, {bucket2}, ...]>	JSON array of bucket objects. See bucket table.

Required Group Bucket Properties

Buckets are represented as JSON objects.

Key	Value	Notes
bucket_id	<number>	Defines ordering of buckets from lowest ID to highest.
bucket_weight	<number>	Can be hexadecimal (with leading 0x) or decimal.
bucket_watch_group	<number> <reserved_group>	Can be hexadecimal (with leading 0x) or decimal. See reserved group table.
bucket_watch_port	<number> <reserved_port>	Can be hexadecimal (with leading 0x) or decimal. See reserved port table.
bucket_actions	<list of actions> <action1=value1,action2=value2,...>	See action table.

Example Group Entry

An example group might look like:

```
{
        "switch" : "00:00:00:00:00:00:00:01",
        "entry_type" : "group",
        "name" : "group-mod-1",
        "active" : "true",
        "group_type" : "select",
        "group_id" : "1",
        "group_buckets" : [
                {
                        "bucket_id" : "1",
                        "bucket_watch_group" : "any",
                        "bucket_weight" : "50",
                        "bucket_actions":"output=2"
                },
                {
                        "bucket_id" : "2",
                        "bucket_watch_group" : "any",
                        "bucket_weight" : "50",
                        "bucket_actions" : "output=3"
                }
        ]
}
```

Using Static Entry Pusher in Practice

The Static Entry Pusher can be scripted using simple python code to control a network. As an example, set up a simple network in a mininet VM after starting the Floodlight controller. The default topology is one switch (s1) and two hosts connected to it (h2 and h3).

```
 sudo mn --controller=remote,ip=<controller ip>,port=6653
 sudo mn --controller=remote,ip=<controller ip>,port=6653 --switch
ovsk,protocols=OpenFlow13
```

Can Use Python to Push Entries

curl was shown as an example above, but Python and other languages can also be used to create a REST client and leverage the Static Entry Pusher API.

The code below will insert a flow from h2 to h3 and one from h3 to h2 in the mininet network created above.

```
import httplib
import json

class StaticEntryPusher(object):

    def __init__(self, server):
        self.server = server

    def get(self, data):
        ret = self.rest_call({}, 'GET')
        return json.loads(ret[2])

    def set(self, data):
        ret = self.rest_call(data, 'POST')
        return ret[0] == 200

    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200

    def rest_call(self, data, action):
        path = '/wm/staticentrypusher/json'
        headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
            }
        body = json.dumps(data)
        conn = httplib.HTTPConnection(self.server, 8080)
        conn.request(action, path, body, headers)
        response = conn.getresponse()
        ret = (response.status, response.reason, response.read())
        print ret
        conn.close()
        return ret

pusher = StaticEntryPusher('<insert_controller_ip')

flow1 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_1",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "active":"true",
    "actions":"output=flood"
    }

flow2 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_2",
    "cookie":"0",
    "priority":"32768",
    "in_port":"2",
    "active":"true",
    "actions":"output=flood"
    }

pusher.set(flow1)
pusher.set(flow2)
```

To test this out, run a pingtest in the mininet vm (note: you may want to disable the learning switch and other routing code in advance to make sure your static entries are taken).

```
mininet> h2 ping h3
```

Using the OpenFlow 1.2+ set_field Action

As an example unrelated to the one above, the following demonstrates how to use the Static Entry Pusher to insert flows that modify header fields in OpenFlow 1.2+:

```
flow3 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_3",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "eth_type":"0x0806"
    "active":"true",
    "instruction_apply_actions":"set_field=arp_tpa->10.0.0.2,output=2"
    }
flow4 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_4",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "eth_type":"0x0806"
    "active":"true",
    "actions":"set_field=arp_tpa->10.0.0.2,output=2"
    }
```

flow3 and flow4 are equivalent. The former uses the proper instruction to apply the actions immediately in an OpenFlow 1.1+ sense. The latter also works; the "actions" key is interpreted as "instruction_apply_actions" if supplied for an OpenFlow 1.1+ switch. Note the use of "set_field" to set the target protocol address of matching ARP packets to 10.0.0.2.