zof Documentation

Release

William W. Fisher

Jan 30, 2018

Contents:

CHAPTER 1

ZOF: OpenFlow App Framework

zof is a Python framework for creating asyncio-based applications that control the network using the OpenFlow protocol. *zof* uses a separate *oftr* process to terminate OpenFlow connections and translate OpenFlow messages to JSON.

There is no built-in OpenFlow API. You construct OpenFlow messages via YAML strings or Python dictionaries. Incoming OpenFlow messages are generic Python objects. Special OpenFlow constants such as 'NO_BUFFER' appear as strings.

An OpenFlow application may be composed of multiple "app modules". The framework includes built-in "system modules" that you can build upon.

1.1 Requirements

- Python 3.5.1 or later
- oftr command line tool

1.2 Install - Linux

```
# Install /usr/bin/oftr dependency.
sudo add-apt-repository ppa:byllyfish/oftr
sudo apt-get update
sudo apt-get install oftr

# Create virtual environment and install zof.
python3.5 -m venv myenv
source myenv/bin/activate
pip install zof
```

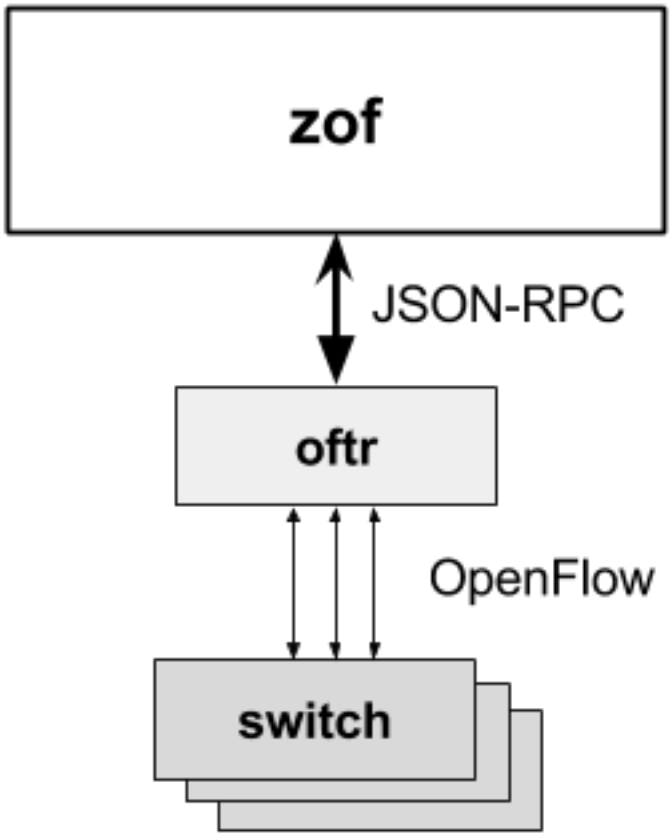

Fig. 1.1: Architecture: The oftr process translates OpenFlow to JSON.

1.3 Demos

To run the controller demo:

```
python -m zof.demo.layer2 --help
```

CHAPTER 2

Event Dispatch

Events are dispatched to apps with the highest precedence first.

All apps are initially sorted by precedence value, highest to lowest. The default precedence value for an app is 100. You have the option to set the precedence value when you call zof.Application. Where two apps have the same precedence value, they remain in their original order.

Events are dispatched one at a time to each app in order. Each app checks its handlers from top to bottom, and runs the first handler that matches. Then, the dispatcher proceeds to the next app.

After the event is dispatched to all apps, the dispatcher yields any running async tasks before processing the next event.

2.1 Event

There are two types of events: message events and internal events. Message events have a 'type' attribute. They represent incoming OpenFlow messages.

```
{
  'type': 'PACKET_IN',
  'msg': {
      ...
  }
}
```

Internal events have an 'event' attribute instead of a 'type' attribute. Internal events are used by the framework for communication between apps.

```
{
  'event': 'SIGNAL',
  'signal': 'SIGHUP',
  'exit': True
}
```

Event attributes can be accessed using dot notation "event.type" or as keys "event['type']".

2.2 Handler

A handler is a function with a *@app.message* or *@app.event* decorator. The function must take exactly one *event* argument.

The decorator describes the handler's trigger condition. @app.message(type, ...) specifies an OpenFlow message type. @app.event(event, ...) specifies an internal event type.

2.3 Handler Functions

A handler function may be synchronous or asynchronous. If a handler is synchronous, the entire handler executes before proceeding to the next app. If the handler is asynchronous, a new async task is created and the *first step* of the async task executes before proceeding to the next app.

```
# Synchronous handler
@app.message('packet_in')
def packet_in(event):
    ...

# Asynchronous handler (uses async def)
@app.message('features_reply')
async def features_reply(event):
    ...
```

Handler functions must not block or perform cpu-intensive operations.

2.4 Message Handlers and Implicit Variables

2.5 Async Tasks and Scope

Use the app.ensure_future() method to start a new asynchronous task. This task is scheduled independently and managed by the framework.

2.6 Blocking or CPU-Intensive Tasks

```
@app.event('start')
async def start(event):
    loop = asyncio.get_event_loop()
    result = await loop.run_in_executor(None, functools.partial(blocking_operation, 1,
↪ foo=2))


def blocking_operation(n, *, foo):
    # Runs in separate thread. Be careful with shared mutable state!
    os.sleep(10)
```

2.7 Stopping Propagation

An app handler can stop propagation of an event by raising a *StopPropagationException*. The handler must be synchronous; calling StopPropagationException from an asynchronous handler will not work.

2.8 Events are Mutable

Events are mutable. An app's handler may modify an event before it reaches later apps. This feature must be used with care, since handlers won't make copies of the event object. See *Signal Handling* for one way event mutability is used.

CHAPTER 3

Scalar Types

3.1 DatapathID

64-bit value uniquely identifying a datapath.

Canonical Type: String

Canonical Form: “hh:hh:hh:hh:hh:hh:hh:hh”

Acceptable Forms: “0x0102” -> “00:00:00:00:00:00:01:02” (String must begin with “0x”)

Reference: *Datapath ID* (OF_v1.5.1: Section 7.3.1)

3.2 VlanNumber

14-bit vlan vid.

0 means no VLAN tag is present. 1-4095 specifies a VLAN tag value. 4096 specifies VLAN tag value of 0.

For compatibility with the OpenFlow spec, 4096-8191 also specify VLAN tags 0-4095.

Negative integers represent a non-zero vlan_vid which does not have the OFPVID_PRESENT bit set.

Canonical Type: Integer

Canonical Form: SInt32

CHAPTER 4

Mixed Types

A mixed type may be either an JSON integer or string. Strings are used for reserved constants in the canonical form.

4.1 PortNumber

32-bit port number.

Canonical Type: Integer or String

Canonical Form: UInt32 | “IN_PORT” | “TABLE” | “NORMAL” | “FLOOD” | “ALL” | “CONTROLLER” | “LOCAL” | “ANY” | “NONE”

CHAPTER 5

Signal Handling

An app handles signals by listening for a SIGNAL event.

```
@app.event('signal', signal='SIGHUP')
def sighup(event):
    event.exit = False
    ...
```

The framework automatically listens for SIGTERM, SIGINT, and SIGHUP signals. By default, these signals cause the program to shutdown cleanly. To prevent the signal from quitting the program, your handler can change *event.exit* to False.

You can also listen for other signals. By default, these signals do not quit the program unless you set *event.exit* to True. To listen for a signal, specify the signal name using the *signal* attribute.

```
@app.event('signal', signal='SIGUSR1')
def siguser1(event):
    ...
```

CHAPTER 6

Reference

6.1 Classes

class zof.**Application**(*name*, *, *controller=None*, *exception_fatal=False*, *precedence=100*, *arg_parser=None*, *has_datapath_id=True*)

The *zof.Application* class represents a controller *app*.

Your app's code registers handlers for events via an *Application* instance.

Parameters

- **name** (*str*) – Name of the app.
- **precedence** (*int*) – Precedence for app event dispatch.
- **exception_fatal** (*bool* | *str*) – If true, abort app when a handler raises an exception. When the value is a string, it's treated as the name of the exception logger *zof.<exc_log>*.
- **arg_parser** (*argparse.ArgumentParser*) – App's argument parser.

name
str – App name.

logger
Logger – App's logger.

args

event(*subtype*, ***kwds*)
Event decorator.

handlers

message(*subtype*, ***kwds*)
Message decorator.

oftr_connection

phase

precedence

class zof.api_compile.**CompiledMessage**
Abstract class representing a compiled OpenFlow message.

_controller
Controller – Controller object.

request(***kwds*)
Send an OpenFlow request and receive a response.

Parameters **kwds** (*dict*) – Template argument values.

request_all(*, *parallelism=1*, ***kwds*)
Send multiple OpenFlow requests and receive responses.

Parameters **kwds** (*dict*) – Template argument values.

send(***kwds*)
Send an OpenFlow message (fire and forget).

Parameters **kwds** (*dict*) – Template argument values.

class zof.api_compile.**CompiledObject**(*controller*, *obj*)
Concrete class representing a compiled OpenFlow object template.

6.2 Functions

zof.**run**(*, *args=None*)
Run event loop for zof.

Parameters **args** (*Optional[argparse.Namespace]*) – Arguments derived from ArgumentParser. If None, use *common_args* parser. If args is a list, pass these to parse_args.

zof.**compile**(*msg*)
Compile an OpenFlow message template.

zof.**common_args**(*, *under_test=False*, *include_x_modules=False*)
Construct default ArgumentParser parent.

Parameters

- **under_test** (*Boolean*) – When true, create an argument parser subclass that raises an exception instead of calling exit.
- **include_x_modules** (*Boolean*) – When true, include "–x-modules" argument.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

C

E

H

L

M

N

O

P

R

S