OFBench: SDN performance test suite on OpenFlow switch

# Introduction

Software Defined Networking (SDN) is an emerging network architecture. The major differ- ence between SDN and the traditional network is the decoupling of the control and data plane. Therefore, in the operation of SDN, the data plane needs to communicate with the control plane through a specific communications protocol. This change brings the operations of the control plane in the foreground as part of the data packet processing. OpenFlow is a major standard for SDN which be managed by Open Network

Foundation (ONF) [1], is a communications protocol used to manipulate the data plane operations by the controller [2, 3, 4]. Therefore, another distinct difference between OpenFlow and the traditional network is the stateful switching rather than the stateless counterpart.

As the packet arrives at the OpenFlow switch, selected tuples in the packet header fields are compared with the rules in the flow table. If there is no match, the packet is treated as a new flow and table-miss operation is triggered. Depending on the configuration, the packet can be either simply dropped or an OpenFlow message is sent to the controller a waiting for further action. The controller may insert a new flow entry in the flow table with action for the matched packet. The flow entry is the record of the data packet, which contains information such as packet headers, counters and instructions [5]. The process of the OpenFlow transaction can be categorized into two parts, one is data-plane-trigger-control-plane (D2C), the other is

control-plane-trigger-data-plane (C2D). The whole process is abbreviated as D2C2D. As such, OpenFlow adds some extra data structures and operations to the data plane. These changes make the OpenFlow data plane (OFD) different from the traditional data plane (TD) .

Currently, there is no performance test standard exist for OpenFlow switch. In the past, the performance test is only focus on the traditional TD part, therefore we propose that the D2C2D process should also be included along with the OFD as part of the testing suite for OpenFlow switch. Cause the packet processing depends on the flow entry which including the some ac- tions. The D2C2D process will affect the flow entry setup when the new type flow coming. Moreover, the OFD process will affect the operation of flow entry in the OpenFlow switch for the existed flows. Due to above mentioned reason, we think the OpenFlow performance testing must be included TD, OFD, D2C, and C2D. The main reason is that the D2C2D process is the core of the operation in both data plane and control plane in the OpenFlow switch. Moreover, the D2C2D process and OFD affect the packet processing in OpenFlow switch. So we think the OpenFlow performance testing must be included TD, OFD, D2C, and C2D.

In our survey, there are some open source tools such as OFTest [6] and Ryu certification [7] exist for the OpenFlow switch conformance test. These tools only determine whether the switch fit the interoperability of OpenFlow protocol or not. As for commercial tools, there is no one except the white paper proposed by Spirent for OpenFlow performance testing [8]. The white paper only addresses some test cases and offers some concepts to evaluate the switch. There are many research literatures address the issues in this topic as well. However, most of them focus on TD testing without addressing the others. The authors in OFLOPS [9] discuss the test of OFD and C2D parts without cover all test matrices of these two parts.

In this thesis, we propose thirteen test cases based on the white paper [8] from Spirent . The test cases, including items such as flow action, pipeline, Packet-in, Packet-out, and Flow-mod, address the most critical parts of OFD, D2C, and C2D testing. Among these thirteen test cases,

there are seven test cases classified as non-trivial. And we are going to focus on the development of test tools for these six out of seven non-trivial test cases.

It is difficult to get the internal parameters from the device-under-test (DUT) for evaluation in these non-trivial test cases due to black-box testing. For example, the action time is hard to measure because the processes in OpenFlow consist of many steps. Normally, we can only measure the end-to-end delay time without exact break down of individual action time where many variables involved . And the idle timeout of flow entry is hard to measure the accuracy by traffic , cause the idle timeout will be reset when packet coming and matched. Therefore

, we propose two methodologies to address these issues. The first one, named as mirror-in- processing, is based on the Apply-Action instruction of OpenFlow to mirror packets in the process. The system is capable of measuring the processing time of each stage in the DUT based on the mirrored packets. So we can inference some exact results. The second one, named as masked entry, is based on the priority, match fields and actions of flow entry to control the timer where two flow entries can be synchronized for proper measurement.

We propose and design an automatic test framework based on controller-agent architecture. The framework is capable of controlling remote agents to generate and analyze networking traffic. The agents provide the analysis results to the controller for test cases. We adopt the white-box test methodology to evaluate and ensure the accuracy of our black-box test cases. In white-box test, we modify and design specific codes in Open vSwitch. The goal is to measure and compare the test results with all test matrices based on the same test steps stated in the black-box test cases. Finally, these test cases are applied to the OpenFlow switches.

The remainder of this work is organized as follows. We introduce the backgrounds of Open- Flow with emphasis on the performance related issues and related works for testing in Chapter

2. The problem statement is provided in Chapter 3. In Chapter 4, we introduce the proposed methodologies for the mirror-in-processing and the masked entry. We further discuss our imple-

mentation detail and experiment results in Chapter 5. Conclusions and future works are finally presented in Chapter 6.

# Background

## OpenFlow

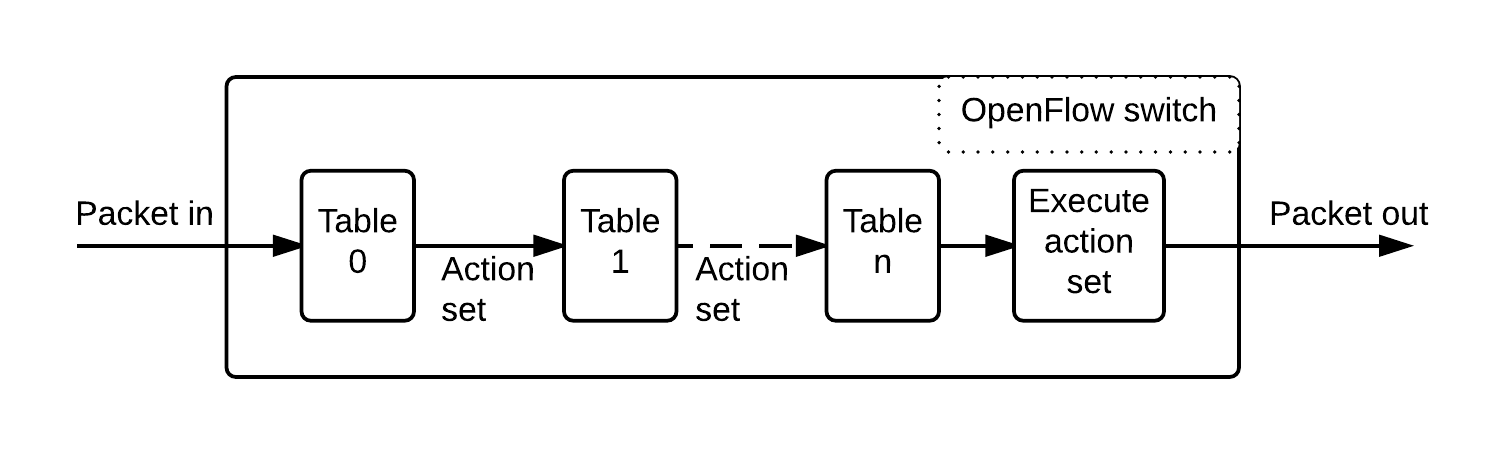
OpenFlow is a communication protocol for SDN, which contains packet processing mech- anism and communications messages and OpenFlow switches. The packet processing mecha- nism is based on flow policy that is a combination of match fields and instruction set that be implemented as flow entry and be stored at flow table. The whole process for packet processing as shown in Figure 1. Firstly, the switch will try to match the flow entries in table 0 when packet coming. Each processing has an action set which records the actions that will be applied to the packet, and default will be empty. When packet matched, the action set will be update and possible be directed to other tables. The directed operation be call pipeline in OpenFlow. When the pipeline processing finished, the action set will be executed.

Figure 1: OpenFlow Pipeline Processing

The flow entry is the most important part of the process which consists of priority, timers, match fields, and instruction set. The priority will affect the executing order of flow entries. The timers contain hard timeout and idle timeout which will trigger removal for flow entry when it expired. The only different is the idle timeout will be reset by packets, but the hard timeout will not be affected. The match fields are a set of records for layer 1 to 4 header for packets. And the instruction set contains operations for packets that be executed when packet matched. The

available instructions and executing order are listed in Table 1.

Table 1: OpenFlow instruction with executing priority

|  |  |  |
| --- | --- | --- |
| Instruction | Description | Executing Priority |
| Meter | Apply the specified metering to packet. | 6 |
| Apply-Actions | Apply actions immediately to packets. | 5 |
| Clear-Actions | Clear action set. | 4 |
| Write-Actions | Write actions to action set. | 3 |
| Write-Metadata | Write the masked metadata value. | 2 |
| Goto-Table | Execute the table pipeline. | 1 |

## OpenFlow performance parameters

There are many messages will be sent from the controller in the C2D part that including the Flow-mod, configuration, and querying the state of the switch. Also, the switch will report some events to the controller in the D2C part that may be Packet-in, Flow-removed, status report, and errors. Theses messages could be the parameter of OpenFlow performance. But in the major situation, the D2C2D process are often be executed for new type flows. The process is consists of the Packet-in, Packet-out, and Flow-mod messages.

For the Packet-in message, the rate generated by OpenFlow switch that will affect the pro- cess of handling the new type flows. If the rate is low, the OpenFlow switch will not able to handle a larger number of new type flows in short time. And the Packet-out will be used to execute the actions for the Packet-in message in the D2C2D process that is processing of the first packet of new type flow normally, it also could be used in topology discovery. The rate of Packet-out determines the capability of packet processing which requested by the controller for OpenFlow switch. Finally, how long the flow entry could active that will be concerned when the switch received the Flow-mod message. So these three messages are the major parameters of OpenFlow performance for the C2D and the D2C parts.

In the OFD part, there are many performance parameters which consist of packet processing and OpenFlow mechanism. The former consist of the table lookup, table pipeline, and time of action set execution. The later are the timeout of flow entry, and the size of flow table.

## Related Work

Currently, there are many solutions focus on TD testing and not addressed D2C, C2D, and OFD except the Spirent and OFLOPS. As Table 2, Spirent proposes the white paper [8] for OpenFlow switch performance testing. It focus on the concept of test cases and give some sug- gestion for each case. Instead the concept, OFLOPS [9] propose two major test cases for the latency of Flow-mod and the execution time of action set. Obviously, it did not cover all of the performance parameters. Opposite, we propose all of the parameters except the latency of Flow-mod and the size of the flow table. Although, the latency could be evaluated by Barrier messages, but it will exist the deviation. There are two reasons, one is that the Barrier messages need the extra time to execute between the controller and the switch. The other is that the im- plementation of Barrier messages in OpenFlow switch may not correctly. The method proposed by OFLOPS [9] which could solve these issues and use the traffic to measure the setup time of multi-flow entries. This way is good enough to evaluate the latency of Flow-mod so we will not cover it. The other exception is the size of the table. For this parameter, it is trivial through basic Flow-mod with an add operation. For Packet-in/out rate we propose the test of buffer size. Both could be evaluated by buffer size test case in the process. The table pipeline, we propose two test cases to get the time and performance for pipeline processing. And the test case for the execution time of action set is more exactly than the method OFLOPS proposed. Finally, we propose a method to verify the accuracy of idle-timeout and hard-time for the timeout of flow entry parameter.

Table 2: Related work comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Parameter | Spirent [8] | OFLOPS [9] | Ours |
| D2C | Packet-in rate | Concept |  | Buffer size |
| C2D | Packet-out rate | Concept |  | Buffer size |
| Latency of Flow-  mod |  | Multi flow entries and traffic  validation |  |
| OFD | Table lookup | Concept |  |  |
| Table pipeline | Concept |  | Time and performance of ta-  ble pipeline |
| Execution time of  action set |  | End-to-end | Exactly action time |
| The timeout of  flow entry | Concept |  | Verify the accuarcy of idle-  timeout and hard-timeout |
| The size of flow  table | Concept |  | trivial |

# Problem Statement

## Notation

Table 3 shows the notations used in this work. Given the switch under test *dut*, the controller *c*, the hosts *H*, the number of tables *N* for *dut*, and the link capacities *CAP* . The performance parameters have three categories, there are the C2D parameters *CD*, the D2C parameters *DC*, and the OFD parameters *Dopenflow* . And the flow entries *F* in *dut* and traffic *T FC* generated by *H* will effect each parameter.

## Problem Description

Using given entities to create a topology and determining the flow entries *F* and the traffic *T FC* to evaluate *CD*, *DC*, and *Dopenflow* . The objective of our work is to assure the most accuracy for each result of *CD*, *DC*, and *Dopenflow* . Due to the black-box testing, the *dut* is not modifiable.

Table 3: Notation Description

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Category | Notation | | | | Description |
| Entity | *c* | | | | The controller |
| *dut* | | | | The switch under test |
| *N* | | | | The number of tables for *dut* |
| *H* = *{hn|n ≥* 2*}* | | | | The set of hosts. |
| *CAP* = *{capc, capn|n ≥* 2*}* | | | | The set of link capacities. *capc*/*capn* is link ca-  pacity between *c*/*hn* and *dut* |
| C2D | *CD* = *{Ppacket−out}* | | | | The set of parameters for C2D. *Ppacket−out*  means the performance of packet-out operation in *dut* |
| D2C | *DC* = *{Ppacket−in}* | | | | The set of parameters for D2C. *Ppacket−in*  means the performance of packet-in operation in *dut* |
| OFD | *Taction−set* | | | | The time of action set execution in *dut* |
| *Ttable−pipline* | | | | The time of table pipeline in *dut* |
| *buf* | | | | The size of buffer in *dut* |
| *Acctimeout* | | | | The accuracy of timeout in *dut* |
| *Ppipeline* | | | | The performance of table pipeline in *dut* |
| *Dopenflow* = *{Taction−set,*  *Ttable−pipeline, buf, Acctimeout, Ppipeline}* | | | | The set of parameters for OFD |
| Process | *T B* = *{tablei|*0 *≥ i < N }* | | | | The set of flow tables in *dut* |
| *F* = *{flowi,j|*0  *N, j >* 0*}* | *≥* | *i* | *<* | The set of flow entries. *flowi,j* mean the flow  entry in *tablei*. |
| *T FC* = *{tfcy|y >* 0*}* | | | | The set of traffics which be sent from *src* to *dst*.  *src ∈ H, dst ∈ H, tfcy* = *{pktz|z >* 0*}*. |

**Example**

Figure 2 shows the parameters under the testing. The pipeline process contains the time and performance of table pipeline (*Ttable−pipeline, Ppipeline*), and the time of action set execution *Taction−set* in *dut*. The set of flow entries *F* determine the pipeline how to work. Each flow entry *flowi,j* has two timers for hard-timeout and idle-timeout, that the accuracy of timeout *Acctimeout* will evaluate each timer correctly expired or not. And combine the *F* and traffic *T FC* to make *dut* to generate the Packet-in messages to *c* or to operate the Packet-out messages sent from

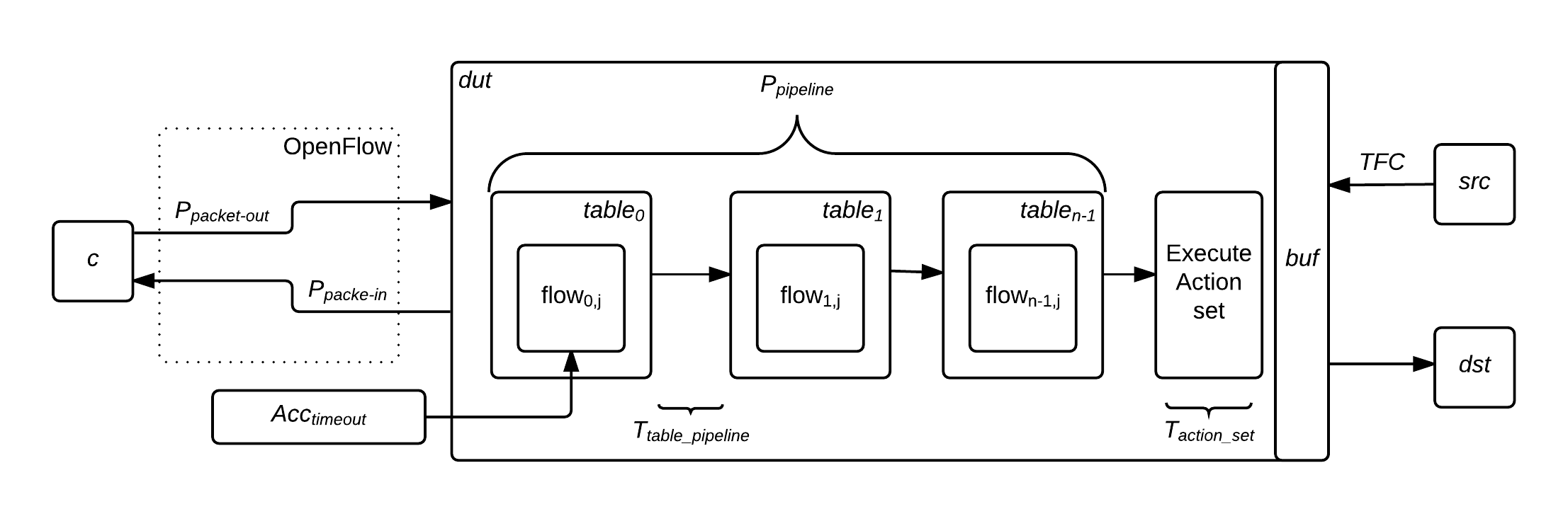
*c*. According the process in *dut* to determine the rate of Packet-in *Ppacket−in* and Packet-out *Ppacket−out*. The Packet-in operation in *dut* that it will keep the packet in buffer *buf* if the buffer is not full.

Figure 2: OpenFlow Performance testing example

# References

[1] *Open Network Foundation*. [https://www.opennetworking.org/about/onf-](http://www.opennetworking.org/about/onf-) overview. [2] *Ryu controller*. https://osrg.github.io/ryu/.

[3] *OpenDaylight controller*. [https://www.opendaylight.or](http://www.opendaylight.org/)g/. [4] *Onos controller*. [http://onosproject.org/.](http://onosproject.org/)

[5] “OpenFlow Switch Specification,” vol. 3, pp. 1–164, 2013.

[6] *OFTest*. [http://www.projectfloodlight.org/oftest/.](http://www.projectfloodlight.org/oftest/)

[7] *Ryu certification*. https://osrg.github.io/ryu/certification.html. [8] Spirent, “OpenFlow Performance Testing,”

[9] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, “OFLOPS: An open framework for OpenFlow switch evaluation,” *Lecture Notes in Computer Science (includ- ing subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7192 LNCS, pp. 85–95, 2012.