

國立交通大學

網路工程研究所

碩 士 論 文

OFBench: OpenFlow 交換器效能測試方法

OFBench: Performance test suite on OpenFlow switches

研 究 生：王辰佑

指導教授：林盈達 教授

中 華 民 國 105 年 6 月

OFBench: OpenFlow 交換器效能測試方法
OFBench: Performance test suite on OpenFlow switches

研 究 生：王辰佑

Student : Chen-You Wang

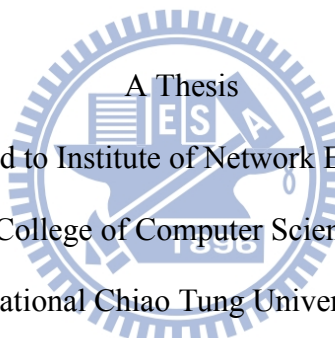
指導教授：林盈達

Advisor : Dr. Ying-Dar Lin

國立交通大學

網路工程研究所

碩士論文

The logo of National Chiao Tung University is a circular emblem. It features a gear-like outer ring. Inside, there is a stylized building or structure with the letters 'ES' and 'A' prominently displayed. The year '1938' is also visible at the bottom of the inner circle.

A Thesis
Submitted to Institute of Network Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science
June 2016
Hsinchu, Taiwan, Republic of China

中華民國 105 年 6 月

學生：王辰佑

指導教授：林盈達

國立交通大學網路工程研究所

摘要

OpenFlow 交換器已在現今網路佔有一定地位，隨之帶來的效能議題也倍受矚目。為此，我們提出五項測試方法供驗證各家 OpenFlow 交換器效能評估，當中的測試指標包含 action time、pipeline time、packet-in rate、packet-out rate、pipeline gain 及 timeout accuracy；此外，為了簡化測試流程，我們設計一自動化測試系統 OFBench，該系統為 controller-agent 架構，並提供測試項目的開發功能。實驗顯示，硬體交換器在 timeout 的實作上較為不足，其實際誤差約為 $\pm 20\%$ ，但在 Packet-in 測試項目中，硬體交換器相對於軟體交換器約有 20 倍的效能差異，另外，在 table pipeline 測試項目中，硬體交換器在 1Gbps 的流量下擁有約 40 – 60% 的 pipeline gain 且其實際可使用的 table 數量僅有 60 多張。而在實驗過程中，我們額外發現交換器在部分功能實作上有 3 項非預期的行為，其一，交換器在 Apply-Action 功能並沒有實作完全；其二，交換器在處理大量 Packet-in 流量時，會導致斷線需重新進行連線；最後，交換器在處理 idle-timeout 時，會有未正確將該值重置，導致 flow entry 提早到期。

關鍵字：軟體定義網路、OpenFlow、效能、交換器、測試

OFBench: Performance test suite on OpenFlow switches

Student: Chen-You Wang

Advisor: Dr. Ying-Dar Lin

Department of Computer and Information Science

National Chiao Tung University

Abstract

Currently, there are many OpenFlow switch products available in the market. The performance issues of OpenFlow protocol are drawing a lot of attentions. In this work, we propose five test cases to evaluate six performance metrics: action time, pipeline time, packet-in rate, packet-out rate, pipeline gain, and timeout accuracy. The switch can be evaluated based on these metrics. And we also propose the automatic test framework: OFBench, which is a controller-agent architecture allowing the development of test case based on high-level script language. The evaluation results reveal a $\pm 20\%$ skewness in idle-timeout accuracy in the hardware switches. For the Packet-in rate, the hardware switches is capable of generating Packet-in frames in the rate ranging from 3000 to 7000 Packet-in per second. The rate is 20 times faster than that generated by the software switches. As for the test of the gain of table pipeline, the hardware switches reach 40 – 60% pipeline gain under the 1Gbps traffic loading. And the switches only have around 60 usable tables. Furthermore, we observed three issues for switch during the testing. Firstly, the switches may not be well implemented on the design of Apply-Action instructions. Secondly, the switches suffer from random crashes with the high volume of bursty Packet-in traffic. Lastly, the timer of idle-timeout is not reset properly when the flow entry matched.

Keywords: SDN, OpenFlow, Performance, switch, testing

誌謝

這篇論文能夠完成，首先要感謝林盈達教授、賴源正教授及賴裕昆教授的費心指導。在每次的討論中，教授們總是不辭辛勞的給予意見，讓我從中能釐清思考的盲點，自己也從中學習到如何進行嚴謹有邏輯的思考，也萬分感謝林盈達教授及賴裕昆教授在修定論文上所花費的時間及心思；此外，還要感謝交大網路測試中心（NBL）所提供的資源及協助，使我能順利完成我的研究。

在兩年的研究生涯中，也要感謝所有高速網路實驗室的所有成員，不論是在學業、生活或研究所給予的支持、鼓勵及建議，透過大家此彼教學相長，使得自己在這段時間頗有一番收穫。

最後感謝我的家人，感謝你們長久下來的關心及照顧，讓我能專注於學業上，並完成學業。



王辰佑 謹致於
2016 年 6 月 3 日

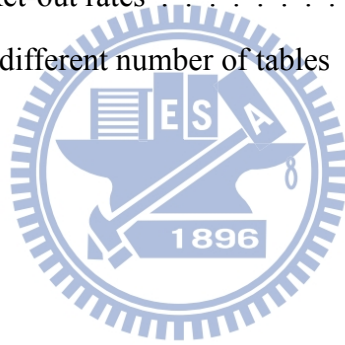
Contents

List of Figures	v
List of Tables	vi
1 Introduction	1
2 Background	4
2.1 OpenFlow	4
2.2 OpenFlow performance parameters	5
2.3 Related Work	6
3 Problem Statement	8
3.1 Notation	8
3.2 Problem Description	8
4 Methodology	10
4.1 Mirror-in-processing	10
4.2 Calculated traffic	12
4.3 Masked entry	14
5 Numerical Results	16
5.1 Implementation	16
5.2 Experiment setup	16
5.3 Experimental Results	17
6 Conclusions	23
References	25



List of Figures

Figure 1 : OpenFlow pipeline processing	4
Figure 2 : OpenFlow performance testing example	9
Figure 3 : Mirror-first-then-action	12
Figure 4 : Mirror-first-then-pipeline	12
Figure 5 : Burst-until-loss	14
Figure 6 : Back-to-back-traffic	14
Figure 7 : Idle-timeout-derived-by-hard-timeout	15
Figure 8 : OFBench architecture	16
Figure 9 : Action Time	18
Figure 10: Packet-in and Packet-out rates	20
Figure 11: Pipeline gain with different number of tables	21



List of Tables

Table 1 : OpenFlow instruction with executing priority	5
Table 2 : Related work comparison	7
Table 3 : Notation description	9
Table 4 : Approach overview	10
Table 5 : OS and hardware specifications for components	17
Table 6 : Hardware switch specifications	17
Table 7 : Buffer size with different frame size for 1Gbps traffic	20
Table 8 : Timeout accuracy	22



Chapter 1 Introduction

Software Defined Networking (SDN) is an emerging network architecture. The major difference 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 is managed by Open Network

Foundation (ONF) [1], is a communications protocol used to manipulate the data plane operations by the controller. 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 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 [2]. 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 the OpenFlow switch. In the past, the performance test only focus on the traditional TD part. However, we think 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 some actions. 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, 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 [3] and Ryu certification [4] 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 [5]. The white paper only addresses some test cases and offers some concepts to evaluate the switch. There are many research works of literature address the issues in this topic as well. However, most of them focus on TD testing without addressing the others. The authors in OFLOPS [6] discuss the test of OFD and C2D parts without covering all test matrices of these two parts.

In this thesis, we propose thirteen test cases based on the white paper [5] 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 the exact breakdown 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 the 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.

The remainder of this work is organized as follows. We introduce the backgrounds of OpenFlow 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 implementation detail and experiment results in Chapter 5. Conclusions and future works are finally presented in Chapter 6.



Chapter 2 Background

2.1 OpenFlow

OpenFlow, a communications protocol for SDN architecture, consists of communications messages and mechanisms enabling packet processing for OpenFlow switches and controllers. The packet processing mechanism is based on flow policy, which is a combination of match fields and instruction sets. The flow policy is implemented as flow entry and stored in the flow table. The whole process for packet processing is illustrated in Figure 1. As the packet arrives, the switch tries to match the flow entries in the table of multiple stages. For each matching process, an action set (with the default of empty) of the particular action is applied to the packet. Once the packet is matched with the entry in the table, the action set will be updated and possibly directed to other tables in a pipelined fashion. Finally, the action set will be executed at the end of the pipeline process.

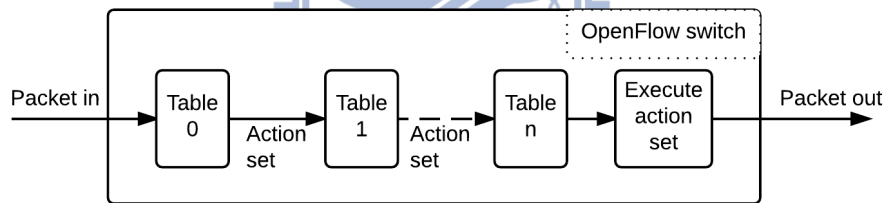


Figure 1: OpenFlow pipeline processing

The flow entry, which is the most important part of the process, consists of priority, counters, match fields, timeouts, cookies, and instructions. The priority affects the executing order of flow entries. The timeouts contain timers of hard-timeout and idle-timeout values for flow entry removal when expired. The match fields are a set of records consist of packet headers ranging from layer one to four. And the instruction set contains operations to be executed for the incoming packet which is matched with the match fields.

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

2.2 OpenFlow performance parameters

In the past, the performance testing focuses on the TD part which includes latency, loss, buffer size, throughput, jitter. However, the most important factor of performance testing for OpenFlow switch should be the interactions between the control plane and the data plane. And considering the extra structures or mechanisms for OpenFlow, we summarize the OpenFlow performance parameters with C2D, D2C, and OFD parts.

In the C2D part, there are many messages, which include the Flow-mod, configuration, and state query of the switch, sent from the controller to the switches. The switches report messages of some events back to the controller in the D2C part. These messages consist of Packet-in, Flow-removed, status report, and errors. Therefore, these messages could be treated as the important parameters for OpenFlow performance measurement. Moreover, in most of the cases, the D2C2D process are often executed for the new type of flows. The process is consisting message transactions of the Packet-in, Packet-out, and Flow-mod.

The rate of Packet-in message generated by OpenFlow switch is an important capability indicator handling the new type of flows. If the rate is low, the OpenFlow switch will not be able to handle a large number of new flows in a short period of time. As for the Packet-out message, it is used to execute the actions for the Packet-in message in the D2C2D process for the first packet of a new flow. The rate of Packet-out message, which is generated by the controller, reflects the capability of packet processing for OpenFlow switch. Finally, the duration of how long the flow entry remains active is set by the Flow-mod message sent to the switch. Therefore, these three messages are the major parameters of OpenFlow performance measurement for the C2D and the D2C parts.

In the OFD part, we summarize the performance parameters in two categories: the packet

processing and OpenFlow mechanism. The former consists of the table lookup, table pipeline, and time of action set execution. The later contains the timeout of flow entry and the size of the flow table.

2.3 Related Work

Currently, there are some testing solutions developed with the focus on TD. Bianco [7] and Emmerich [8] provide the measurement methodologies for the forwarding throughput and packet latency of switch in underloaded and overloaded situations. Gelberger [9] proposes the cost measurement which includes throughput, latency, and jitter for switching and routing on OpenFlow. Jarschel [10] propose the model based on forwarding speed and blocking probability to estimate the packet sojourn time.

Recently, Spirent proposes a white paper [5] for OpenFlow switch performance testing. It focuses on the general concepts of testing and provides suggestions for each test cases. More specifically, the authors in OFLOPS [6] propose two major test cases for the latency measurement of Flow-mod and the execution time of action set.

Although the latency can be evaluated by the Barrier messages, however, the measurement result is not accurate where deviation may occur. There are two major reasons. The one is that the Barrier messages need extra time to be processed between the controller and the switch. The other is that the process of Barrier messages in OpenFlow switch may not be implemented correctly. The methods proposed by OFLOPS [6] solve these issues by using the traffic to measure the setup time of multi-flow entries. In advance, the authors in OFLOPS propose the OFLOPS-Turbo [11] which enhances the framework by 10 Gbps Ethernet platform to evaluate the next-generation OpenFlow switch. However, the system does not address any methodologies for switch testing in detail.

Due to the fact where latency of Flow-mod may be varied under different testing situations, Handfield [12] finds out the major impact factors which include the priority, current flow table size, and the operation of Flow-mod. Bianco [7] and Tanyingyong [13] estimate the effects of table lookup for hash and linear flow. For the throughput, both of them [7][13] find out the hash flow have better performance. For the processing time, Bianco [7] estimate it for hash and

linear flows with different table size. The results are constant and independent of the table size for both types. The author of High-fidelity switch models [14] proposed a model to emulate the vendor hardware switches including the Flow-mod rate without stating the methodologies.

Obviously, as summarized in Table 2, the existing works of literature fail to cover all of the performance parameters needed to evaluate the OpenFlow switch under test. Therefore, in this thesis, we propose how to measure all of the major performance parameters except the latency of Flow-mod and the size of the flow table.

For the size of the flow table measurement, we can simply use a basic Flow-mod with an add operation. For the measurement of Packet-in and Packet-out rate, we propose to evaluate them by buffer size test case in the process. For the measurement of table pipeline, we propose two test cases to obtain the time for evaluating the pipeline processing performance. The proposed test case for the execution time measurement of the action set is more accurately than the method proposed by OFLOPS [6]. Finally, we propose a method to verify the accuracy of idle and hard timeout for the flow entry

Table 2: Related work comparison

Type	Parameter	Spirent [5]	OFLOPS [6]	Bianco [7]	Handfield [12]	Ours
D2C	Packet-in rate	Concept	N/A	N/A	N/A	Buffer size
C2D	Packet-out rate	Concept	N/A	N/A	N/A	Buffer size
	Latency of Flow-mod	N/A	Traffic validation	N/A	Impact factors	N/A
OFD	Table lookup	Concept	N/A	Constant	N/A	N/A
	Table pipeline	Concept	N/A	N/A	N/A	Time, pipeline gain
	Execution time of action set	N/A	End-to-end	N/A	N/A	Exactly action time
	The time-out of flow entry	Concept	N/A	N/A	N/A	Accuracy
	The size of flow table	Concept	N/A	N/A	N/A	Trivial

Chapter 3 Problem Statement

3.1 Notation

Table 3 shows the notations used in this work. Given the switch under test dut , the number of tables for dut is denoted by N . The controller, the hosts, and the link capacities are denoted by c , H and CAP respectively. The performance parameters consist of three categories: they are the C2D parameters CD , the D2C parameters DC , and the OFD parameters $D_{openflow}$. And the flow entries F in dut and traffic TFC generated by H will affect each parameter.

3.2 Problem Description

The topology is created based on the given entities shown in Table 3. The flow entries and traffic are determined based on the parameter of F and the TFC to evaluate CD , DC , and $D_{openflow}$. The objective of our work is to assure the accuracy of each measurement result of CD , DC , and $D_{openflow}$. Due to the black-box testing, the dut is not modifiable.

Example

Figure 2 shows the parameters of the test. The pipeline process contains the time and performance of table pipeline ($T_{table-pipeline}$, P), and the time of action set execution $T_{action-set}$ in dut . The set of flow entries F determines the operation of the entire pipeline. Each flow entry, denoted by $flow_{i,j}$, has two timers of hard-timeout and idle-timeout. The accuracy of the timeout, denoted by $Acc_{timeout}$, evaluate whether each timer expired correctly or not. With the combinations of F and traffic TFC the dut is able to generate the Packet-in messages to c or to process the Packet-out messages sent from c . The rate of Packet-in and Packet-out for a given dut is determined by the parameters of PIR and POR . The buffer size of dut denoted by buf . If buf is remaining, dut stores the TFC into the buffer and raises the Packet-in operation with an identity when the new flows arrival. And the c generates the Packet-out operation with the corresponding identity to dut , so dut could forward the new flows.

Table 3: Notation description

Category	Notation	Description
Entity	c	The controller
	dut	The switch under test
	N	The number of tables for dut
	$H = \{h_n n \geq 2\}$	The set of hosts.
	$CAP = \{cap_c, cap_n n \geq 2\}$	The set of link capacities. cap_c/cap_n is link capacity between c/h_n and dut
C2D	$CD = POR$	The parameter for C2D. POR means the throughput of packet-out operation in dut
D2C	$DC = PIR$	The parameter for D2C. PIR means the throughput of packet-in operation in dut
OFD	$T_{action-set}$	The time of action set execution in dut
	$T_{table-pipeline}$	The time of table pipeline in dut
	buf	The size of buffer in dut
	$Acc_{timeout}$	The accuracy of timeout in dut
	P	The gain of table pipeline in dut
	$D_{openflow} = \{T_{action-set}, T_{table-pipeline}, buf, Acc_{timeout}, P\}$	The set of parameters for OFD
Process	$TB = \{table_i 0 \leq i < N\}$	The set of flow tables in dut
	$F = \{flow_{i,j} 0 \leq i < N, j > 0\}$	The set of flow entries. $flow_{i,j}$ mean the flow entry in $table_i$.
	$TFC = \{pkt_z z > 0\}$	The traffic which be sent from src to dst . $src \in H, dst \in H$.
	T_{table_lookup}	The time of looking up the flow table in dut .
	T_{apply_action}	The time of executing Apply-Action in dut .
	$T_{idle-timeout}, T_{hard-timeout}$	The timeout value for idle/hard timeout flow entry.
	$T_{idle-expired}, T_{hard-expired}$	The arrival tiem of flow-removed message for idle/hard timeout flow entry at c .
	$T_{idle-duration}, T_{hard-duration}$	The duration of flow-removed message for idle/hard timeout flow entry.

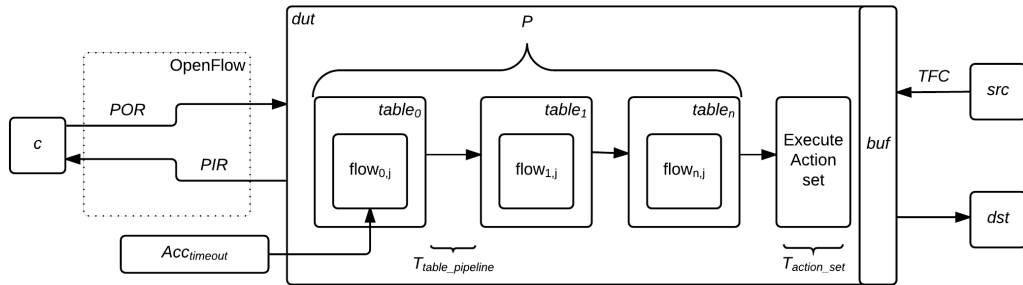


Figure 2: OpenFlow performance testing example

Chapter 4 Methodology

The measurement of performance parameters for the proposed five test cases is listed in Table 4. These cases are classified into three categories: Mirror-in-processing, Masked entry, and Calculated traffic. And each case has a corresponding named method to its spirit.

Table 4: Approach overview

Category	Test Case	Method	Output
Mirror-in-processing	Action set time	Mirror-first-then-action	$T_{action-set}$
	Pipeline Time	Mirror-first-then-pipeline	$T_{table-pipeline}$
Calculated traffic	Buffer size	Burst-until-loss	buf, PIR, POR
	Pipeline performance	Back-to-back-traffic	P
Masked entry	Timeout accuracy	Idle-timeout-derived-by-hard-timeout	$Acc_{timeout}$

4.1 Mirror-in-processing

In the OpenFlow performance testing, it is difficult to get the test values and results directly. However, by leveraging on the mirror operation with Apply-Actions instruction within the processing, we can duplicate the packets before some OpenFlow operations for measurement and comparison.

Mirror-first-then-action

The purpose of this test case is to measure the execution time of action set in OpenFlow switch. In OpenFlow protocol, there are many kinds of actions which include forwarding, packet modification, and tagging operation. The packet modification may cause longer operation time in some switches. Therefore, the processing latency of action is used for major performance benchmark.

In the normal case, the time of action denoted by $T_{action-set}$ can be calculated by measuring the end-to-end latency. Since there are many variables affecting the testing result, we propose the mirroring concept to get the start time of action set for better measurement accuracy. As

shown in 3, the pkt_z is denoted as the mirrored packet. The value of α denotes the arrival time of pkt_z and β be the arrival time of pkt_z' . We derive the action set execution time as

$$T_{action-set} = \beta - \alpha. \quad (1)$$

Mirror-first-then-pipeline

The purpose of this case is to measure the time of table pipeline in OpenFlow switch. The mirroring methodology can be applied to this test case as well. This is because the instructions of table pipeline are executed after Apply-Actions.

As illustrated in Figure 4, we assume the pipeline stage consists of two tables: $table_0$ and $table_1$. And the flow entries are almost the same except the flow entry in $table_0$ has GoTo instruction. The time of T_1 includes the time of table lookup and Apply-Actions operations in $table_0$. The time of T_2 includes the time of $T_{table-pipeline}$, table lookup, and Apply-Actions operations in $table_1$. Due to the same match fields and actions of flow entry in $table_0$ and $table_1$, the processing latencies, denoted by T_{table_lookup} and T_{apply_action} respectively, are the same in $table_0$ and $table_1$. By considering the Round-Trip Time (RTT), that we can derive the time of table pipeline as follows.

$$T_{table-pipeline} = T_2 - T_1 + RTT/2 \quad (2)$$

Where the T_1 is

$$T_1 = T_{table_lookup} + T_{apply_action} + RTT. \quad (3)$$

And the T_2 is

$$T_2 = T_{table-pipeline} + T_{table_lookup} + T_{apply_action} + RTT/2. \quad (4)$$

The RTT can be derived by $RTT = (pkt_z/cap_n) * 2$.

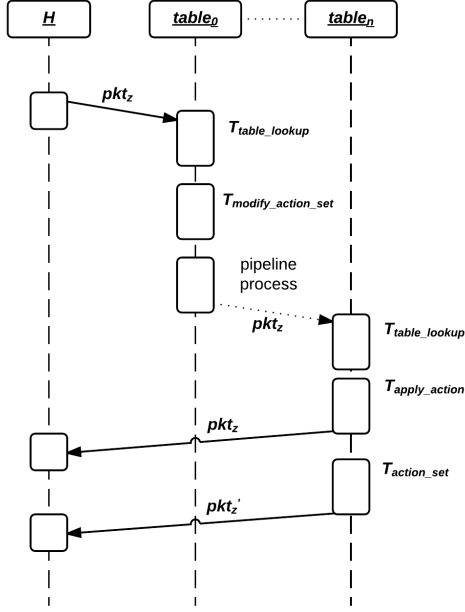


Figure 3: Mirror-first-then-action

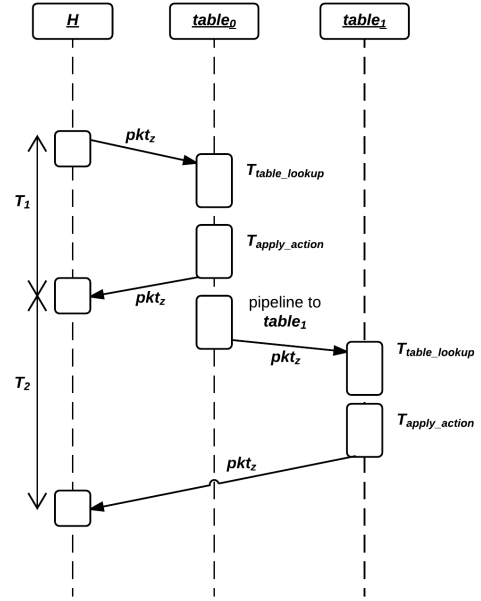


Figure 4: Mirror-first-then-pipeline

4.2 Calculated traffic

Burst-until-loss

The purpose of this test case is to measure the buffer size, the throughputs of Packet-in and Packet-out. The main reason why we combine the measurement of these three parameters into one case is that the Packet-in and Packet-out messages may depend on the size of the buffer in dut.

As shown in Figure 5, the *dut* has the flow entry that matches the *TFC* and sends the Packet-in message to the controller *c*. The controller *c* will keep all Packet-in messages and generate the expected Packet-out messages after the traffic generating from host *H* is done. We assume there is no packet loss occurred at *c* and all Packet-in messages consume the entire buffer space in the dut. When the condition $m < n$ occur, there exists packet loss at the dut. Therefore, we can derive the buffer size, denoted as *buf* in the following formula.

$$buf = m * pkt \quad (5)$$

The throughput of Packet-in can be derived by the following formula.

$$PIR = \text{the number of Packet-in}/T_1 \quad (6)$$

And the throughput of Packet-out can be derived by the following formula.

$$POR = m/T_2 \quad (7)$$

Back-to-back-traffic

The purpose of this test case is to measure the gain of table pipeline in *dut*. As illustrated in Figure 6, the latency of *TFC* covers total processing time for $n+1$ tables. In each table, packets are matched with the flow entry and sent to the next table in a pipeline fashion. The pipeline gain of *pkt* processing time at each table can be derived by the packet processing time and idle times. We denote the processing time of *pkt* and the idle time as T_{pkt} , T_{bubble} respectively. And we derive the pipeline gain from the following formula.

$$P = T_{pkt}/(T_{bubble} + T_{pkt}) \quad (8)$$

Where the T_{pkt} is the processing time of the packet at each table. Due to the difficulty of measuring the T_{pkt} in black-box testing, we assume an uniform processing times of *pkt* at each table for the performance evaluation. Therefore, we approximate the T_{pkt} by using the processing time of the first packet T_{init} based on the following formula.

$$T_{pkt} \approx T_{init}/(n+1) \quad (9)$$

In equation 8, the time of $T_{bubble} + T_{pkt}$ reflect the total time of *pkt* processing time and the idle time for a single table. In our measurement, we known the total number of packets $n+1$ and the total duration T . Thus, assuming there are n packets sent after the first packet, the average time of $T_{bubble} + T_{pkt}$ can be derived by the following formula.

$$T_{bubble} + T_{pkt} \approx (T - T_{init})/n \quad (10)$$

The T_{init} and T are the processing time for the first packet, and the total processing time for $n+1$ packets respectively. We denote the α_z as the time of sending pkt_z and the β_z as the arrival time of pkt_z . And the T_{init} and the T be derived by the $T_{init} = \beta_1 - \alpha_1$ and $T = \beta_{n+1} - \alpha_1$ respectively.

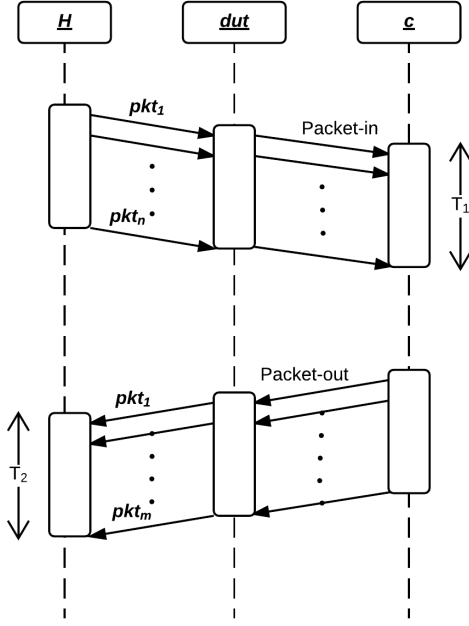


Figure 5: Burst-until-loss

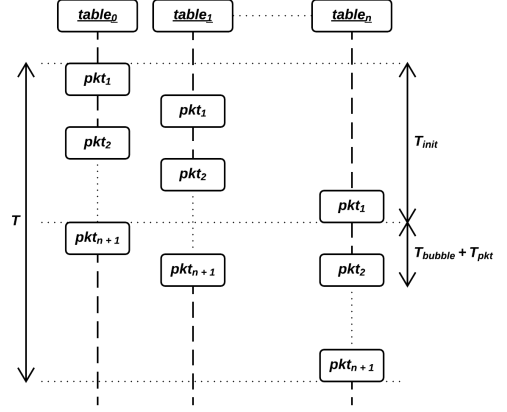


Figure 6: Back-to-back-traffic

4.3 Masked entry

Base on the definition of OpenFlow specification, the flow table can have multiple entries with the same match fields of different priority.

Idle-timeout-derived-by-hard-timeout

The purpose of this case is to measure the accuracy of timeout for a given flow entry. The timeout measurement includes hard-timeout and idle-timeout. For the measurement of hard-timeout, we can evaluate the accuracy simply based on the packet injected into the DUT. But for the measurement of idle-timeout, the deviation of the accuracy could be larger due to the inter-arrival time of the traffic as the flow entry being matched.

To evaluate the accuracy of idle-timeout, we propose to use the masked entry to avoid the reset of the timer for idle-timeout at the wrong time by the arriving packet. In Figure 7, there are two flow entries added into the *dut*. The first one is the idle-timeout flow entry and the second is the hard-timeout flow entry which is the masked entry with higher priority. Both of the flow entries are matched with the *TFC* and the time of idle-timeout ($T_{idle-timeout}$) equals the

time of hard-timeout ($T_{hard-timeout}$). When the idle-timeout flow entry is added to the *dut*, the timer for idle-timeout stays zero as the packets are forwarded based on the matched entry. After the setting of the idle-timeout flow entry, the hard-timeout flow entry is added to the *dut*. At this moment, the hard-timeout flow entry will take over the idle-timeout flow entry because of its higher priority. And the timers for both of the flow entries will start counting from zero as they are synchronized. Finally, there are 2 situations illustrated in Figure 7a and Figure 7b. In Figure 7a, the hard-timeout flow entry is expired early than idle-timeout flow entry. It means that there exists the skewness for the timer of idle-timeout. In order to evaluate skewness, we increase the $T_{hard-timeout}$ and set the hard-timeout flow entry again. The process stated above is repeated until the idle-timeout flow entry is expired early as illustrated Figure 7b. According to the testing processes, we can derive the accuracy of idle-timeout based on the following formula.

$$Acc_{idle-timeout} = T_{idle-duration} - T_{idle-expired} - T_{idle-timeout} \quad (11)$$

The $T_{idle-duration}$, and $T_{idle-forward}$ are alive and active times for idle-timeout flow respectively. We denote the α_z as the arrival time of pkt_z . The $T_{idle-forward}$ can be derived from the following formula.

$$T_{idle-forward} = \alpha_n - \alpha_1 \quad (12)$$

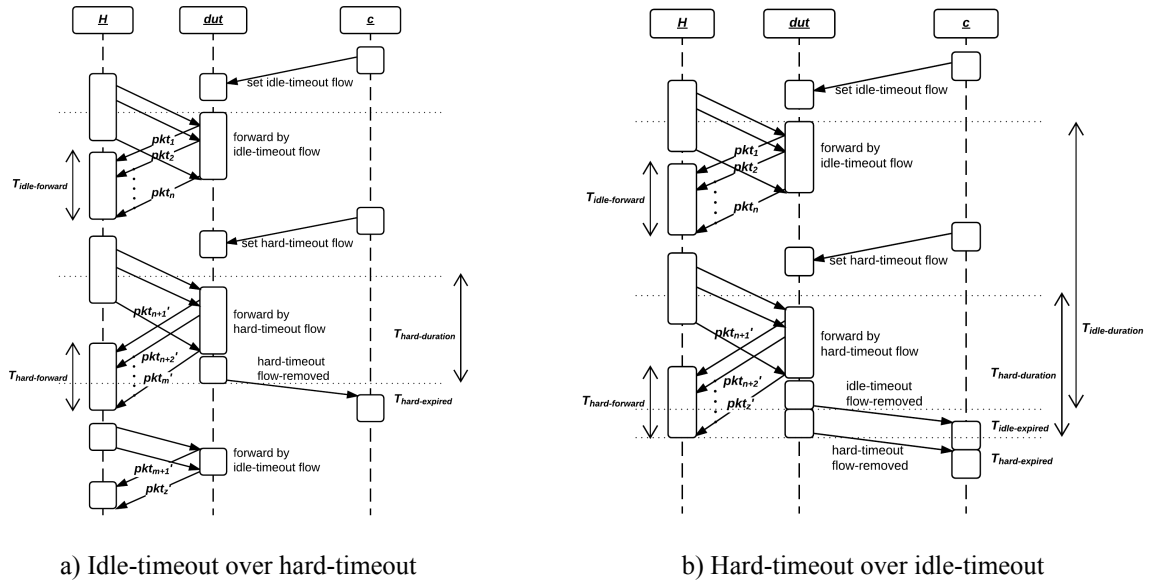


Figure 7: Idle-timeout-derived-by-hard-timeout

Chapter 5 Numerical Results

5.1 Implementation

We propose OFBench, an automatic testing framework integrated with Ryu controller¹ and Ostinato traffic generator². The architecture of OFBench, shown in Figure 8, is a controller-agent architecture. Controlled by the controller, each agent is capable of generating traffic, capturing and parsing packets. The test cases of OFBench are written in scripts for basic OpenFlow operations and test results analysis.

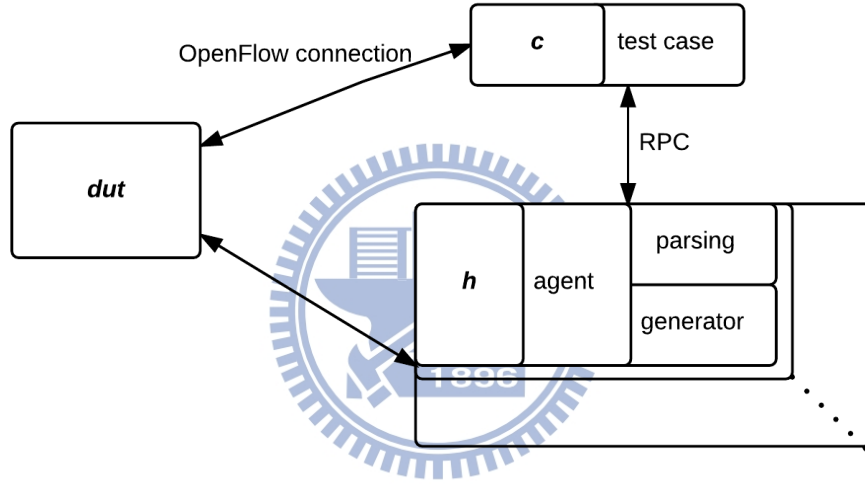


Figure 8: OFBench architecture

5.2 Experiment setup

Five test cases are developed with test scripts for the OFBench framework. The available configurations of the test cases consist of frame size, transmission rate, and some specific values. The available frame sizes are 64, 128, 256, 512, and 1024 bytes, and the available transmission rates are ranging from 64 Kbps to 1Gbps. In our experiment, we use UDP traffic for the testing of all test cases. We conduct the experiments with three physical PCs. The operation systems

¹Ryu, version 4.0, <https://osrg.github.io/ryu/>

²Ostinato, version 0.7.1, <http://ostinato.org/>

and specifications of those PCs are shown in Table 5.

Table 5: OS and hardware specifications for components

Component	Core	Clock rate	OS	Kernel
Controller	4	3.3GHz	Ubuntu 14.04	4.2.0-27
Host	4	3.2GHz	Ubuntu 14.04	4.2.0-27
Switch	2	3.1GHz	Ubuntu 14.04	3.13.0-24

We use the above environment to test four DUTs. The DUTs details are shown in Table 6.

Table 6: Hardware switch specifications

Switch	CPU	Core	Clock rate	Memory	Buffer	OS version
Pica8 P-3290	MPC8541	1	1 GHz	512 MB	4 MB	v2.6.1
Pica8 P-3297	P2020	2	1.33 GHz	2 GB	4 MB	v2.6.1
Edge-corE AS4610-30T	ARM Cortex A9	2	1 GHz	2 GB	N/A	v2.6.4
Centec V350	e500v2	1	533 MHz	2 GB	N/A	v3.1(11), 1.alpha

5.3 Experimental Results

In this section, we present our five performance metrics: action time, pipeline time, packet-in rate, packet-out rate, pipeline gain, and timeout accuracy. These metrics try to distinguish the switches implementation for features.

Each metrics are evaluated based on the proposed five test cases. And the statistic results are collected with five rounds of testing.

Action Time

We try to find out the relationship of action time with different combinations of actions. The tested actions are based on a set of packet modifications ranging from layer two to four. The test case is conducted based on six different actions of packet modification. The results of action time are shown in Figure 9. The six modifications of packet are ordered by source IP address, destination IP address, source Ethernet address, destination Ethernet address, UDP source port

and UDP destination port respectively. For each testing scenario, we generate traffic at the rate of 1,000 packet-per-second (pps) with a total of 10,000 packets.

Based on the results shown in Figure 9, the result for hardware switch Centec V350 is non-available due to the mirror and action does not execute properly. For the result, The software switch has the lower action time which ranging from 2 to 3 microseconds. However, the action times for hardware switches have at least 100 microseconds. If the implementation of action is pure hardware based, the action time possible be nano seconds. But these results and the CPU specification are in direct proportion. According to the specifications, the Open vSwitch has the highest clock rate, Pica8 P-3297 followed, Pica8 P-3290 and Edge-corE AS4610 final. Thus, we think the implementation of action is software based on switches.

Otherwise, the variation of action time is small due to the different number of modifications. Thus, we conduct this test case on white box environment. We use the kernel trace tool (ftrace) to record the execution time of `set_ip_addr` function for Open vSwitch. The result of white box measurement approximates 0.6 microseconds. And the first modification action has the highest time cost. This result explains the reason for the small variation.

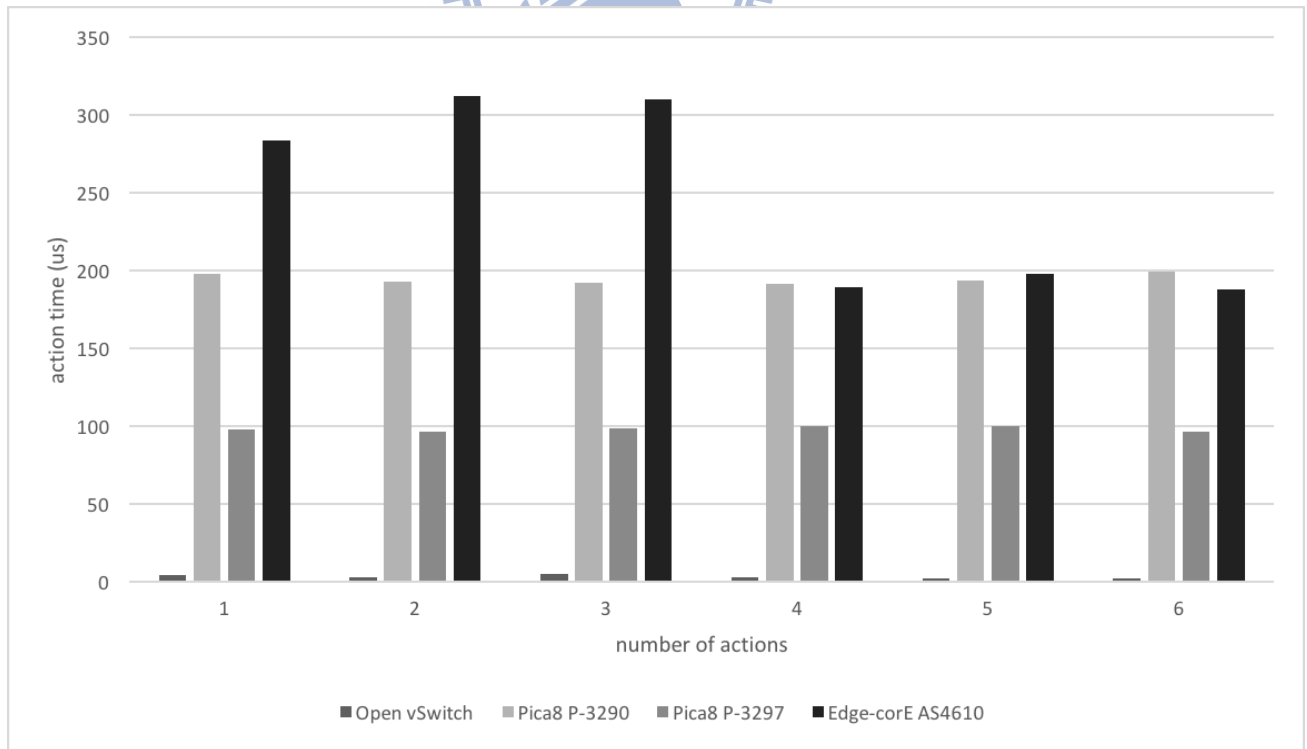


Figure 9: Action Time

Pipeline Time

For our pipeline time test case, the results are unavailable for Pica8 switches due to the Apply-Action is not working properly. The hardware switches can not mirror packets between the pipeline process. Thus, we only have the result for Open vSwitch with different frame sizes ranging from 64 to 1024 bytes. The pipeline time measured is in the range of 1.1 to 2 microseconds with the positive correlation of frame size.

Buffer Size and Packet-in/out

For the buffer size test case, we try to measure the Packet-in rate, Packet-out rate, and buffer size within one test. Therefore, we fix the transmission rate of 1Gbps with different frame sizes for the measurement of Packet-in rate. The results of Packet-in and Packet-out rate are shown in Figure 10. The results be collected at the beginning of the testing are discarded. This is mainly because the traffic generator takes five to ten seconds for stabilizing the testing traffic. According to our results, the rate of Packet-in for Open vSwitch is approximately 20 times lower than that of the hardware switches under test. It indicates the software switch has poor handling capability with the small frame size for Packet-in operation.

For the Packet-out rate, the Pica8 switches have poor handling capability with the small frame sizes, especially the 64 bytes. However, the measurement results for Open vSwitch are different. It is working well on the Packet-out operation with the frame size of 64 bytes. And the throughputs of Packet-out for Open vSwitch are better than those observed in other switches under test.

The measurement results of buffer size for switches under test are shown in Table 7. There are some error messages generated from the DUTs during the testing. The errors, identified as buffer unknown for Packet-out messages lead to the failure of Packet-out operation. The test results indicating packet loss occurred at the DUTs. Finally, we mark the buffer size of the DUT is not available (N/A) when the number of errors is greater than the number of Packet-out messages sent from the controller.

In Table 7, only the buffer sizes tested with frame size of 64-byte are shown. For other frame size, the buffer sizes of the switches are not available. We suspect the anomaly behavior

of DUTs is due to the CVE-2016-2074 [15] issues for the Open vSwitch implementation.

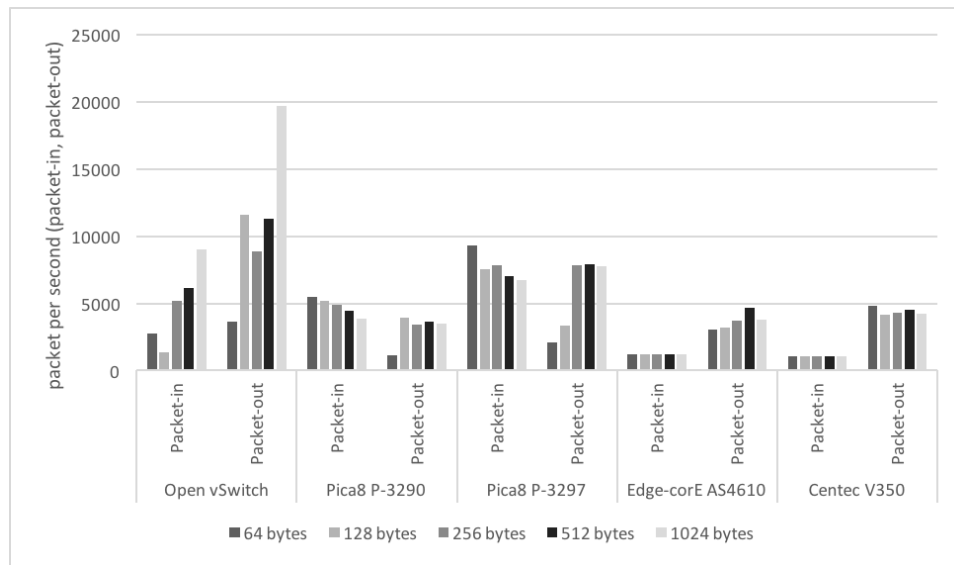


Figure 10: Packet-in and Packet-out rates

Table 7: Buffer size with different frame size for 1Gbps traffic

Frame size(bytes)	Open vSwitch	Pica8 P-3290	Pica8 P-3297	Edge-corE AS4610	Centec V350
64	13504	8256	16192	32768	342464
128	N/A	N/A	N/A	N/A	N/A
256	N/A	N/A	N/A	N/A	N/A
512	N/A	N/A	N/A	N/A	N/A
1024	N/A	N/A	N/A	N/A	N/A

Pipeline Gain

For the gain of table pipeline, the test case is conducted to go through from 20 to 254 tables. with transmission rates 1024 Mbps and the duration is 10 seconds. Considering the warm up for traffic generator, we use the masked entry to filter out the traffic from 0 to 5 seconds. The results are shown in Figure 11, the software switch have poor performance on multiple table pipeline. For hardware switches, the number of tables is inversely proportional to pipeline gain. This indicates the hardware switches have the larger idle time when multiple tables pipelining. Otherwise, all switches have the black-hole situation on 70 tables which indicates the number

of available tables for switch just only 60 tables. And the results for Centec V350 switch is not available due to the switch only have a single table.

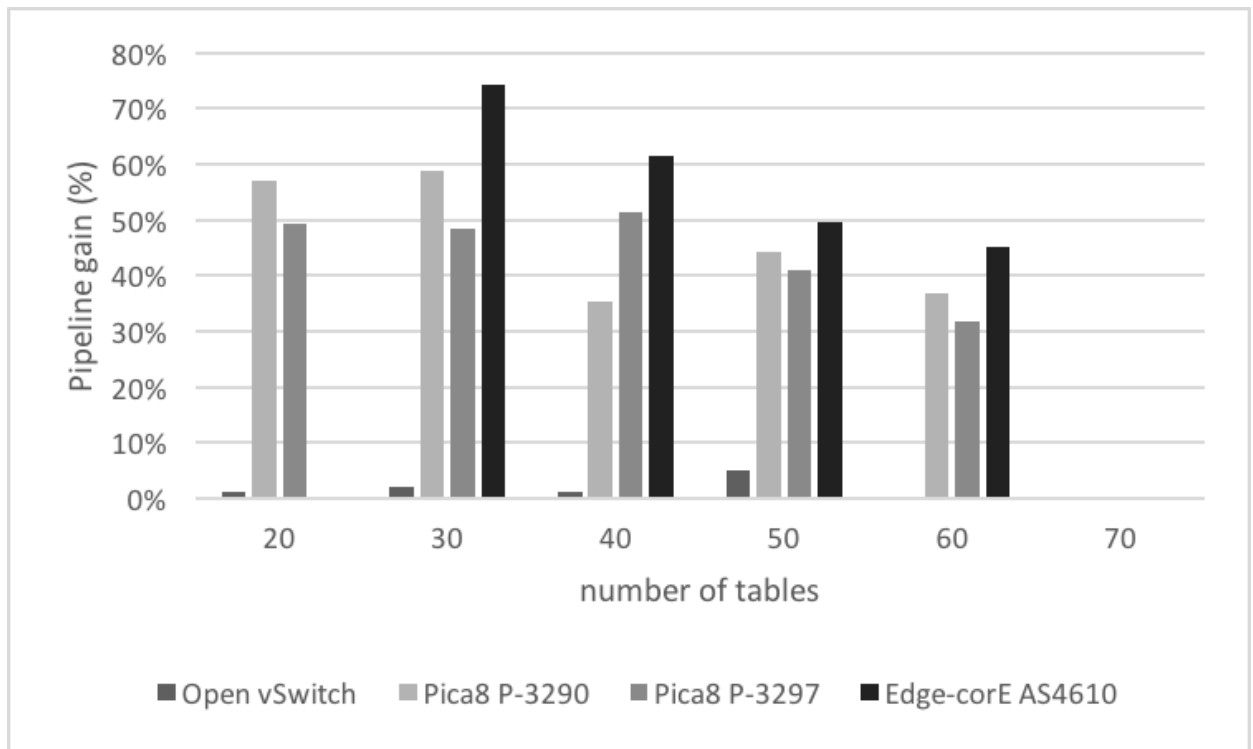


Figure 11: Pipeline gain with different number of tables

Timeout Accuracy

In the testing of timeout accuracy, we set the idle-timeout to five seconds and vary the hard-timeout from three to seven seconds. The traffic is generated at the rate of 10K pps with 64 bytes frame size. The results are shown in Table 8. The deviation of idle-timeout for hardware switch is around $\pm 20\%$. In our test case, we make the switch to swap idle-timeout and hard-timeout flow entries. This operation may cause the switch can not properly handle the idle-timeout properly, and result in the larger deviation. Eventually, the Pica8 switches may not reset the timer for idle-time properly when the idle-timeout flow entry matched.

Table 8: Timeout accuracy

Switch	$Acc_{idle-timeout}$	$Acc_{hard-timeout}$
Open vSwitch	2%	0%
Pica8 P-3290	-16.24%	4.7%
Pica8 P-3297	24.64%	3.3%
Edge-corE AS4610	10%	0%
Centec V350	17.64%	1.66%



Chapter 6 Conclusions

In this work, we focus on the test of OpenFlow switch performance based on the following categories: control-plane-trigger-data-plane, data-plane-trigger-control-plane, and OpenFlow data plane. We propose five test cases to evaluate six performance metrics: action time, pipeline time, packet-in rate, packet-out rate, pipeline gain, and timeout accuracy. These six items can be used to evaluate and reflect the switch performance.

Larger idle timeout deviation for hardware switches

In the results of timeout accuracy, the hardware switches have around $\pm 20\%$ deviation for idle timeout. This may cause the error for applications based on the idle-timeout.

Better new flow handling capability for hardware switches

In the testing results, the hardware switches are capable of generating Packet-in rate in the ranging from 3000 to 7000 Packet-in per second. This rate is approximately 20 times higher than that of the software switches.

Well pipeline implementation for hardware switches

For the measurement of pipeline gain, The hardware switches reach 40 – 60% pipeline gain under handling the 1 Gbps traffic. There has a large gap between hardware and software switches. And the number of available tables have the large deviation. All the switch under test, these switches only have around 60 usable tables, but these switches said they have 254 tables in the feature report.

Issues for switch implementation

We observed issues and problems during the testing for the OpenFlow switched. Firstly, the switches may not be well implemented on the design of Apply-Action instructions. Secondly, the switches crashed with the high volume of burst Packet-in traffic. Moreover, the timer of idle-timeout is not reset properly when the flow entry matched.

Future work

For our experiment, performance factor affected by the loading of the switched is excluded. This may cause some variance for our testing results presented. We plan to apply this factor and discuss some advanced issues to our experiment in the near future. Furthermore, the calculation of the queuing time will be integrated in the pipeline time. In the measurement of pipeline time, we plan to use the finished time at the first table instead of the sending time of the packet. Thus, the measurement of pipeline performance should be more accurate.



References

- [1] *Open Network Foundation*. <https://www.opennetworking.org/about/onf-overview>.
- [2] “OpenFlow Switch Specification,” vol. 3, pp. 1–164, 2013.
- [3] *OFTest*. <http://www.projectfloodlight.org/oftest/>.
- [4] *Ryu certification*. <https://osrg.github.io/ryu/certification.html>.
- [5] Spirent, “OpenFlow Performance Testing,” 2015.
- [6] 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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7192 LNCS, pp. 85–95, 2012.
- [7] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, “OpenFlow switching: Data plane performance,” *IEEE International Conference on Communications*, 2010.
- [8] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, “Performance characteristics of virtual switching,” *2014 IEEE 3rd International Conference on Cloud Networking, CloudNet 2014*, pp. 120–125, 2014.
- [9] A. Gelberger, N. Yemini, and R. Giladi, “Performance analysis of Software-Defined Networking (SDN),” in *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, pp. 389–393, 2013.
- [10] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, “Modeling and performance evaluation of an OpenFlow architecture,” *2011 23rd International Teletraffic Congress (ITC)*, pp. 1–7, 2011.
- [11] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. W. Moore, “OFLOPS-Turbo: Testing the next-generation OpenFlow switch,” *IEEE International Conference on Communications*, pp. 5571–5576, 2015.

- [12] R. B. Handfield and K. McCormack, “What You Need to Know About SDN Flow Tables,” *Supply Chain Management Review*, no. September, pp. 29–36, 2015.
- [13] V. Tanyingyong, M. Hidell, and P. Sjödin, “Improving PC-based OpenFlow switching performance,” *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, pp. 8–9, 2010.
- [14] D. Y. Huang, K. Yocum, and A. C. Snoeren, “High-fidelity switch models for software-defined network emulation,” *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, p. 43, 2013.
- [15] *CVE-2016-2074: MPLS buffer overflow vulnerabilities in Open vSwitch.*
<http://openvswitch.org/pipermail/announce/2016-March/000082.html>.

