Configuring and Controlling a Software Defined Network with a Pica8 Switch

Jason Loux, Dawei Li, and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, USA
{tuf43782, dawei.li, jiewu}@temple.edu

***Abstract*—Current network protocols are exceedingly strict, making the administrative duties of network providers difficult. The switches that control a network rely on stringent definitions, usually defined by the company who designed the switch, with which they can forward packets. Software Defined Networking (SDN) is a concept which allows control over the operations of a switch via remote software applications. But just how much control can we exhibit while writing these applications? And how well does a particular switch respond to this control? Using a Pica8 3297 OpenFlow switch in conjunction with five PowerEdge R210 severs and a Cisco switch, we analyze the response time and the control that the switch maintains while controlled by the Ryu SDN framework.**

***Index Terms*—OpenFlow, Pica8, Software Defined Networking (SDN), Ryu**

I. Introduction

Nothing will hurry the industrial and commercial acceptance of SDN faster than concrete examples which demonstrate its capabilities on an obtainable switch. Pica8 3297 is such a switch, with OpenFlow, a new open standard integrated into many routers and switches. OpenFlow's key feature is that it separates the control plane (that which controls hardware) from the data plane (packets). [1] Current switches and/or routers without OpenFlow try to combine the two planes making it difficult to exhibit direct control. [2] By focusing more on the control aspect of the switches in a network, we are able to make it easier to perform administrative actions on all switches in a network.

The overhead for this control is carried out by the software Ryu, an open-source SDN management controller. [3] It is the perfect software to tackle this job because of its mild learning curve in which applications can be written, and its capabilities with many physical OpenFlow switches.

We also had configure our network's servers and switches with particular network interfaces. (See Fig.1) Then our physical environment could be considered for testing.

II. Test Scenarios

1) Response Time of Switch When Adding a Flow: We analyze how quickly the switch responds to an 'add flow' action by the controller to generate the turnaround time for this response. We can also observe how efficient our particular management controller is in carrying out these actions.

2) Handling of Parallel Packet Transfer: In this example of parallel packet transfer, two servers will attempt to send a large amount of data to one server at the same time. The goal of this test is to see how well the switch handles traffic going to one server. We will also gain a better understanding of how the Ryu management framework controls the switch and of how efficient it is in displaying this control.

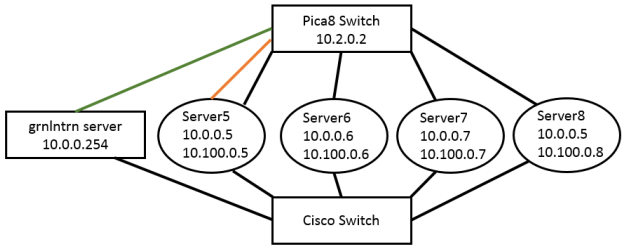

Fig. 1. Note that the grnlntrn server is connected to the Pica8 via a network management port. Server5 is connected to Pica8 via the console management port, giving us direct access and control to the switch to the switch if we are logged-on to Server5. The IP Addresses used in this diagram depict the interfaces used for testing. The first IP is the first interface; the second IP is the second interface.

3) Bandwidth Control using Flow Queueing: With queueing, a controller can set up and configure data structures which map flows to a specific queue. [5] Therefore, we can configure a specific flow to enter a queue which distributes data with whatever limitation we would like. In this way we can implement bandwidth control by setting a queued flow to a port. We can then monitor how effectively adjustments are made by the controller to fluctuate this bandwidth speed.

III. Procedures

A. Adding a Flow via Ryu Controller Application

1) Run server script on Server5. 2) Activate Ryu management application on grnlntrn to begin controlling of switch via software. 3) Once the controller states that it is connected and that preliminary packets are sent through without error, activate client script on Server6. 4) Timestamp when the flow is added via grnlntrn which ensures connection between sending and receiving servers. 5) Timestamp when Server5 actually succeeds in its connection.

B. Parallel Packet Processing Control

This test will have two parts and respecitvely the procedures are as follows:

1) Activate listening servers to send information, in this case, servers 5 and 7. 2) Run Ryu application, which will install and delete flows every three seconds for each sending port. In this way, every three seconds each server will get a chance to send their information. 3) Activate retrieval of first and second files, which will both run as background processes.

1) Activate the same listening servers again. 2) Run a basic Ryu application which will install the necessary flows to allow connection between all servers. 3) Run a python application which simultaneously retrieves the data from both servers and time stamps when both files have completed data transfer.

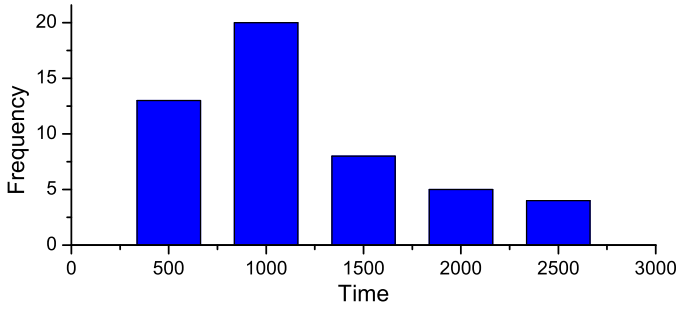

Fig. 2. Above is a histogram of the time between which the controller added the flows and when the sending server noticed it was connected. This gives us an understanding of how quickly the Ryu framework is able to install flows onto the switch and of how quickly the switch implements these flows.

C. Bandwidth Control using Flow Queueing

1) Activate listening servers to wait to send data until a connection is made (Servers 5 and 7). 2) Run Ryu application which, once Server6 attempts to make a connection, will install a queued flow from port 3 (Server7) to port 2 (Server6) quartering port 3's bandwidth (to roughly 250 megabits). 3) Run Server6's client code so connection is attempted to both servers 5 and 7. 4) Use iperf and wireshark to monitor bandwidth of Server7. 5) Install a new flow which restores Server7's bandwidth. 6) Analyze Server7's connectivity and ensure bandwidth is restored to one gigabit.

IV. Results

Adding a Flow via Ryu Controller Application

After fifty tests across the network, we found that it took an average of 0.912 milliseconds for the switch to respond to an added flow via the controller application (See Fig.2). This time is expensive, which leads us to believe that a C-language based controller is the most effecient choice for large networks. We can see that this result creates significant overhead when we use the turn-by-turn algorithm in the next test.

Parallel Packet Processing Control

First, we analyze the time it took for each individual file to complete data transfer to server six. The first file, from server five, averaged about 20.37 seconds. From server seven, the second file took 21.23 seconds. It is expected that the first file completed its transfer, considering that first set of flows installed were always the ones which allowed connection from Server5 to Server6. This seems pretty reasonable for a 1.1 gigabyte file, however, this was not the aggregate time it took for both files finish; that time averaged at about 23.1 seconds (See Both Non-Parallel[NP] in Table I).

TABLE I. Parallel Processing Results (ms)

File1	File2	Both [NP]	Both [P]
20368	21233	23098	21473

In the second part, the only time which was measured was the time that it took for both files to finish transferring their data. This time averaged around 21.47 seconds (Both Parallel[P] in Table I); this is the aggregate time it took for both files to transfer their data (1.625 seconds less than the aggregate time in the prior test).

Bandwidth Control using Flow Queueing

As data is sent from both servers, Fig.3(a) illustrates that the packet lengths for Server5 going to Server6 are clearly larger than the packets going from server7 to server6. This is as expected, considering in the first half of this test the bandwidth on server7 was quartered.

(a)

Source	Destination	Protocol	Length
10.100.0.6	10.100.0.5	TCP	66
10.100.0.5	10.100.0.6	TCP	17442
10.100.0.6	10.100.0.5	TCP	66
10.100.0.7	10.100.0.6	TCP	1514

(b)

Source	Destination	Protocol	Length
10.100.0.7	10.100.0.6	TCP	1514
10.100.0.7	10.100.0.6	TCP	17442
10.100.0.6	10.100.0.7	TCP	66
10.100.0.7	10.100.0.6	TCP	1514
10.100.0.7	10.100.0.6	TCP	8754
10.100.0.7	10.100.0.6	TCP	10202
10.100.0.6	10.100.0.7	TCP	66
10.100.0.6	10.100.0.7	TCP	66
10.100.0.7	10.100.0.6	TCP	5858
10.100.0.6	10.100.0.7	TCP	66
10.100.0.7	10.100.0.6	TCP	14546

(c)

```
[  5] local 10.100.0.7 port 5001 connected with 10.100.0.6 port 56386
[  5]  0.0-10.2 sec   315 MBytes   260 Mbits/sec
[  4] local 10.100.0.7 port 5001 connected with 10.100.0.6 port 56387
[  4]  0.0-10.0 sec  1.10 GBytes   941 Mbits/sec
```

Fig. 3. (a) Note Server5's sent 17442 byte packet size compared to Server7's 1514 byte packet size. This demonstrates the controller's ability to limit bandwidth to a particular port on the Pica8 switch. (b) Note Server7's sent packet size increase when the bandwidth-restoring flow is installed. (c) Here we see the bandwidth of server7 after limiting flow installed, and then once the restoration flow is installed.

In Fig.3(b) we can see the packet length increasing from server7 to server6 as time moves forward, showcasing that the bandwidth increased once the restoring flow was installed. Fig.3(c) provides a summary of the bandwidth control using iperf monitor. The first iperf test was captured while the first limiting flow was installed on server7 and the second after the restoring flow has been installed via the controller.

V. Conclusion

While installing flows onto the switch took under a millisecond to complete, this is still a huge step forward for the SDN community. The amount of time and money that goes into getting new protocols from a switch provider for a switch that does not have OpenFlow is nothing short of disparaging for a network provider. The fact that the Pica8 switch was able to handle parallel packet processing in a port so much faster than a turn-based scheme suggests that it can handle vast amounts of data from multiple sources through its ports at one time. This is great news for any large networking company that will have this demand from a switch at all times. Finally, Ryu's ability to control bandwidth on our physical switch an encouragement for anyone who would like to perform this important administrative task on their network.

References

[1] Open Networking Foundation, *Software-Defined Networking (SDN) Definition*, 2016. [Onlne]. *https://www.opennetworking.org/sdn-resources/sdn-definition*

[2] Open Networking Foundation, *ONF Overview*, 2016. [Onlne]. *https://www.opennetworking.org/about/onf-overview*

[3] Ryu SDN Framework Community, *What's Ryu?*, 2014. [Onlne]. *https://osrg.github.io/ryu/*

[4] Pica8 Inc, *Product Documentation*, 2016. [Onlne]. *http://www.pica8.com/support/documentation*

[5] Brocade Communications Systems Inc, *FastIron Ethernet Switch Software Defined Netwrking (SDN) Configuration Guide*, 2015. [Online]. *http://www1.brocade.com*