Communications and Network, **2013, 5, 434-437**
http://dx.doi.org/10.4236/cn.2013.53B2080 Published Online September 2013 (http://www.scirp.org/journal/cn)

SDN-Based Switch Implementation on Network Processors

Yunchun Li, Guodong Wang
School of Computer Science and Engineering, BeiHang University, Beijing, China
Email: wangguodong2350@126.com

Received June 2013

ABSTRACT

Virtualization is the key technology of cloud computing. Network virtualization plays an important role in this field. Its performance is very relevant to network virtualizing. Nowadays its implementations are mainly based on the idea of Software Define Network (SDN). Open vSwitch is a sort of software virtual switch, which conforms to the OpenFlow protocol standard. It is basically deployed in the Linux kernel hypervisor. This leads to its performance relatively poor because of the limited system resource. In turn, the packet process throughput is very low. In this paper, we present a Cavium-based Open vSwitch implementation. The Cavium platform features with multi cores and couples of hard accelerators. It supports zero-copy of packets and handles packet more quickly. We also carry some experiments on the platform. It indicates that we can use it in the enterprise network or campus network as convergence layer and core layer device.

Keywords: SDN; Open vSwitch; Network Processors; OpenFlow

1. Introduction

Software Defined Network (SDN) is the main approach to achieve network virtualization. The idea was originated at Stanford University Ethane items. The Open Flow [1] protocol is a new standard that provides programming interface on switch or routers. With the new protocol, the control and data planes are decoupled, network intelligence and state are logically centralized, and underlying network infrastructure is abstracted from the applications. This enables enterprises to build highly scalable, flexible networks that readily adapt to changing business needs.

Open vSwitch [2] is developed by Nicira Network and has been widely used because of its great scalability and programmability. Enterprises and carriers can gain unprecedented programmability, automation, and network control. Open vSwitch is deployed in the linux kernel. The kernel bridge is replaced by the kernel module of Open vSwitch. It supports a variety of standard management interfaces, such as NetFlow, sFlow, CLI and so on. And there has been the optimization on the PC platform [4]. But its performance is relatively slow because of the cpu overload. In fact, the performance experiments have been done in [5]. Stanford implements the switching reference on NetFPGA by offloading packet processing from host CPU to NIC [6].

The Cavium [3] network processor provides two runtime modes: SE-S and SE-UM. SE-S mode can achieve better performance based on data plane hardware units, without context switching overhead.

In this paper, we propose to implement the Open vSwitch based on Cavium network processor. We port the kernel module to network processor platform and improve the throughput significantly.

The paper is organized as follows. First, Network Processor platform and Open vSwitch are introduced, followed by the Cavium switch design and implementation. After experiments, the design finally reaches the goal.

2. Open vSwitch Principle of Work

Open vSwitch supports the OpenFlow protocol, which forms the bridge between controller and switch. The communication model is shown in **Figure 1**.

While Open vSwitch can be used as stand-alone switching equipment for packet forwarding, the main mode of operation is switching packets by the command of remote controller. It configures and manages the switch through OpenFlow, and it also provides programming interface to users for enforcing flow rules, modifying flow table and so on. The basic infrastructure is shown in **Figure 2**.

Open vSwitch can run in user space, after loading the

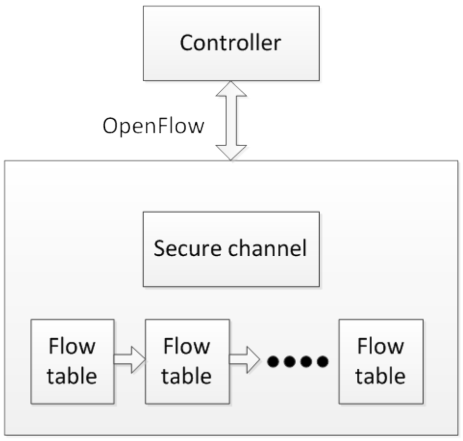

Figure 1. Communication model.

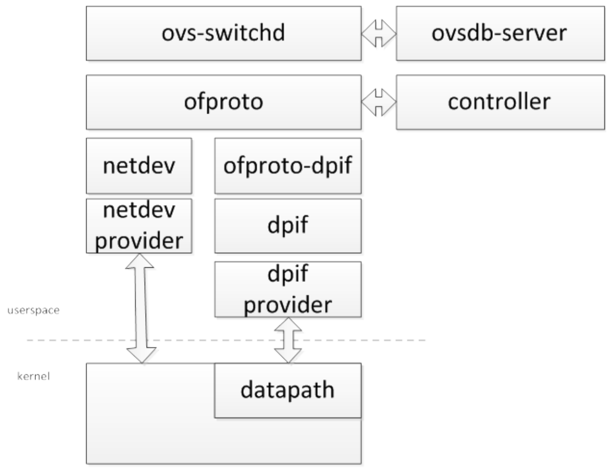

Figure 2. Open vSwitch infrastructure.

kernel module packets can be processed in the kernel and transmitted directly, which is more efficient. As Above figure shows, datapath module is responsible for the lookup and port management. These operations command is passed to the datapath through dpif provider. It is developed using netlink communication. The interaction model is shown in **Figure 3**.

3. Architecture Design

The system is divided into data plane and control plane. Data plane runs in SE-S mode on the platform. It provides the process of table looking up and forwarding packets directly. Control plane is to pass the non-match packets to controller through OpenFlow and notify the datapath the command from controller at the same time. This works in SE-UM mode. The architecture is shown in **Figure 4**.

The control path consists of configuration, secure channel and message passing. Configuration is responsible for configuring connection to controller, MAC learning, VLAN etc. Secure channel process the non-match packet and

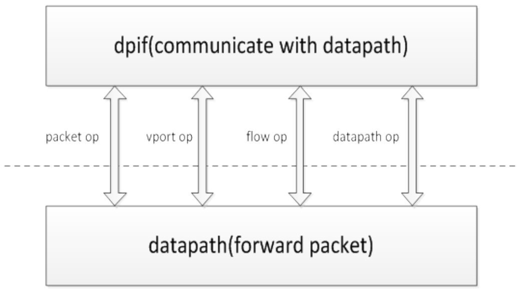

Figure 3. The interaction between kernel and userspace.

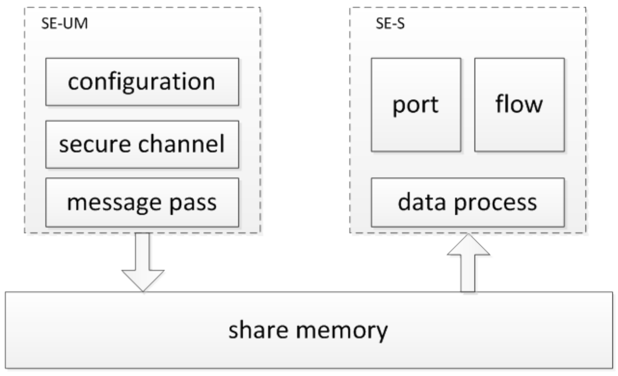

Figure 4. The system architecture.

handle it to controller. The message passing module is designed to send the command to forward path. It facilitates the procedure of making rules from secure channel to datapath. This module is implemented by the share memory provided by the Cavium platform.

The forward path includes management of flow table and port. Flow table module operates the flow table such as updating entry and inserting entry into flow table. Port module manages the ports on platform. It enables the control plane to configure the port and reports the statistical information on it. Data processing module provides the interface between the port and the control plane.

The procedure of datapath is show in **Figure 5**. There are several actions for the target packets: port output, add VLAN field, send to controller and so on. When the packet passes through the port, the port records the information. In addition, we compute the checksum of packet header and inspect the MTU of frame.

4. Open vSwitch Implementation on Cavium

4.1. Zero Copy

Original OVS (Open vSwitch) exchange message through netlink [7] message between user space and kernel space. When a packet did not match with the flow table, datapath call the ovs_dp_upcall function to copy the packet to user space. During the procedure, skb_copy_and_csum_dev function would take a pretty long time.

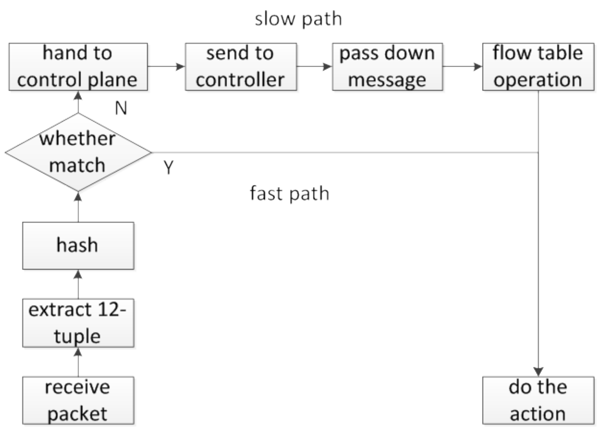

Figure 5. Procedure of incoming packet.

In order to reduce the process time, we take advantage of the platform SSO unit. It defines the packet as a work, which is scheduled in multiple work queues. Every work contains a group info. The group info represents which core the work should be processed on. The core process the packet based on the priority of work. Thus we define two different group numbers for the data plane and control plane respectively. For a non-match work, we assign the group number for control plane. This way reduces the copying time and improves the throughput. The packet process step is shown in **Figure 6**.

4.2. Communication in Share Memory

Cavium platform belongs to SMP architecture. Both SE-UM and SE-S are able to access the **physical** memory directly in the same address space.
Cvmx_bootmem_alloc_named function allocates a name block of memory from the free list. We use this block as the message queue. Control plane puts the message into it and data plane gets the message from it. The message format is shown below:

```
struct sw_cvmx_msg{
      uint8_t cmd;    //command
      uint32_t size; //size of data
      void * data;    //message data
};
```

The data field stores data according to the message command. There are three different types:
dpif_cvmx_datapath, dpif_cvmx_vport and dpif_cvmx_flow.

The control path procedure is as follows:

```
do{
wqe = get_work (non-match work);
if (wqe == NULL) continue;
else
        pass the work to controller;
parse openflow message from controller;
assemble the sw_cvmx_msg;
put the message into queue;
}while (running);
```

The data path procedure is as follows:

```
do{
wqe = get_work();
if (match packet)
            action operation;
else
        assign the group number to control plane
if (has message in queue)
        get message from queue;
}while (running);
```

4.3. Port and flow Table

Port is responsible for forwarding and receiving the packets. IPD/PIP unit on the platform manage the ports through API (such as cvmx_helper_get_ipd_port). In the system we implement the operation for the RGMII interface. Port info is defined in the vport_cvmx structure.

Traditionally, a flow is defined based on 5-tuple: src IP, dst IP, protocol, src port and dst port. However, OpenFlow represents a flow the 12-tuple. Flow entry is defined in the sw_cvmx_flow structure.

5. Experimental Evaluation

5.1. Experiment Setup

It shows that NP-based Open vSwitch realization is indeed better than the PC platform. We carry on the experiment on Cavium CN5860 and x86platform. The setting is shown in **Figure 7**.

We use the SmartBits [8] as the test tool. There are two gigabit ports on the tool. One of them is used to send packet to DUT (Device Under Test), the other port is responsible for receiving packets from DUT output port.

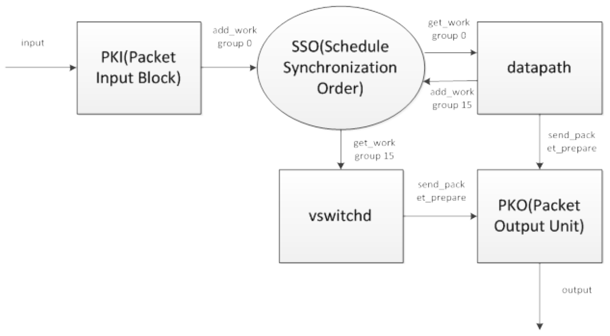

Figure 6. Packet process step.

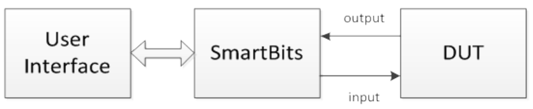

Figure 7. Experiment model.

5.2. Experimetal Scenario

To investigate the performance and scalability aspects of the system, we mainly carry out two experiments as follows:

We generate UDP packets of different sizes (64, 500, 1500 bytes) to test the throughput. The result is shown in **Table 1**.

We measure the throughput in different cores separately (1, 3, 6, 12 cores) to test the scalability. And the result is shown in **Table 2**.

We can make several observations on the results. The NP-based switch performs better than the PC-based switch when forwarding the packets. The data-copying overhead is reduced by using the share memory. And as the core number increases, the performance is definitely improved as well.

6. Summaries

This paper proposed and implemented the NP-based OVS, porting the datapath from kernel space to SE-S mode. The test shows that we improve the performance significantly by utilizing the multi-core processors. However, this article only implements the exact match in the flow table lookup operation. The wildcard match will be carried out in the future work.

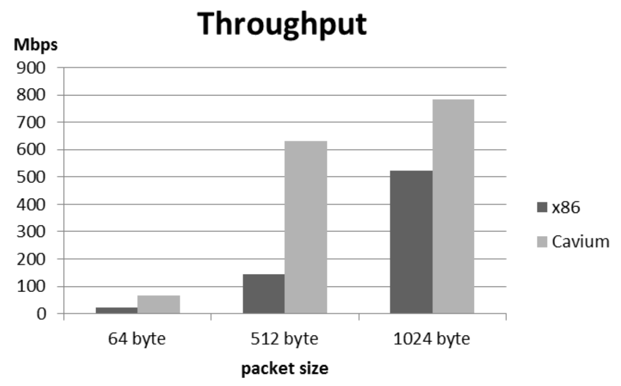

Table 1. Throughput.

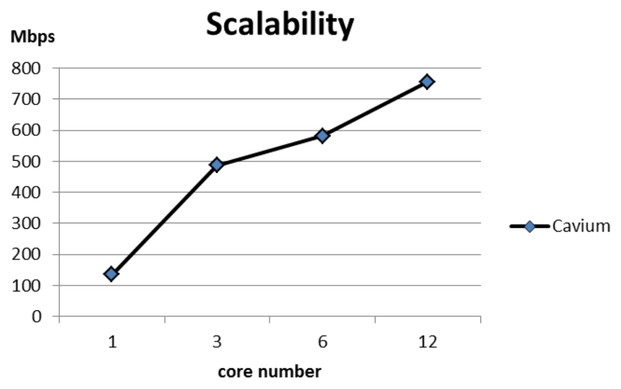

Table 2. Throughput on different cores.

REFERENCES

[1] N. McKeown, T. Anderson and H. Balakrishnan, "OpenFlow: Enabling Innovation in Campus Networks," *Computer Communication Review* (*ACM SIGCOMM*), Vol. 38, No. 2, 2008, pp. 69-74.

[2] K.-K. Yap, T.-Y. Huang, *et al*. "Towards Software-Friendly Networks," *Proceedings of the first ACM Asia-Pacific Workshop on Workshop on Systems*, 2010, pp. 49-54.

[3] The Cavium Website. http://www.cavium.com

[4] V. Tanyingyong, M. Hidell and P. Sjodin, "Using Hardware Classification to Improve PC-Based OpenFlow Switching," *IEEE* 12*th International Conference on High Performance Switching and Routing*, 2011.

[5] A. Bianco, R. Birke, L. Giraudo and M. Palacin, "Openflow Switching: Data Plane Performance," *IEEE International Conference on Communications* (*ICC*), 2010, pp. 1-5.

[6] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow Switch on the Netfpga Platform," *ANCS*: *Proceedings of the* 4*th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008, pp.1-9.

[7] W. Klaus, "Linux Network Architecture: Design and Implementation of Network Protocols in the Linux Kernel," 1972.

[8] SmartBits_Overview_detail_20051008.pdf http://www.spirent.cn/Products/Smartbits