A Comparative Study of Zynq-based OpenFlow Switches in a Software/Hardware Co-design

Jian Kang[†], Xiaojun Hei[†], Jianjian Song[‡]
[†] Huazhong University of Science and Technology, Wuhan, China
[‡] Rose-Hulman Institute of Technology, Terre Haute, IN 47803
Email: {kangjian,heixj}@hust.edu.cn, song@rose-hulman.edu

Abstract*—The end-to-end design principle has been re-examined over the years with the increasing number of middle-boxes on the Internet. The newly released Xilinx Zyqn-based chipsets have been reshaping popular embedded computing platforms, which provide cost-effective but all programmable approaches to enable intelligence at the network edge. In this paper, we applied the hardware/software co-design method, utilized the Zynq-based ONetSwitch board as the hardware platform, and reconstruct the Vivado switch hardware project using TCL scripts. We designed and implemented a software defined networking (SDN) switch based on the Xilinx SDK and its accompanying compiler. We modified and re-compiled the boot file and Linux operating system kernel, installed the transplant Linux operating system to support the OpenFlow switching module, making the platform to achieve software-defined network switching capabilities by modifying the FPGA hardware structure of the software switch for accelerating packet processing and flow-table matching. We constructed an SDN testbed using the re-constructed switch system, the Ryu controller and the client building software. We conducted a series of the high-definition video streaming experiments. The results demonstrated the performance difference between two types of the SDN switches based on the software flow table and the hardware flow table. We utilized network performance testing tools, such as iPerf and Ping, to evaluate the streaming performance including throughput, delay and delay jitter for the SDN switches with the software flow table implementation and the hardware flow table implementation. The experiment results demonstrated the accelerating capability of hardware flow table matching, indicating that the hardware and software co-design method is promising to provide a broad design and optimization space for network systems. As a case study, this switch project established a demonstrating design process, verified the hardware platform and the software tool chain to accomplish a hardware/software co-design. Our experiments have laid a good foundation for further research and development work, and may initiate a path to ultimately realize the Xilinx Zynq based smart edge network with design approaches and methods.

***Index Terms*—Software Defined Networks, OpenFlow, Software/Hardware Co-design, Switch**

I. Introduction

The traditional Internet is constructed based on the TCP/IP protocol suit. The nominal feature of the IP-based network is the distributed control architecture. Each switch or router determines the forwarding of data packets based on their own local routing tables. Whenever the network states change, network state information are exchanged between routers relying on various dynamic routing protocols, and routers re-calculate the routing table. Due to the nature of the distributed control, in-complete or inaccurate network information often leads the network control to be sub-optimal. The rapid development of Internet applications have been driving the network to become more intelligent. The emergence of a large number of middle boxes demonstrates the feasibility of enabling network intelligent at the edge in an evolutionary approach. More and more edge devices need a unified network control and management. Edge devices are of potential to become the communication/information/storage/control center for the smart edge networking scenario, which requires strong control functions and traffic delivery capacity. The software-defined networking (SDN) proposes various features such as the centralized management, the separation of the control plane and the data plane. These features are benefit to build a measurable, manageable, and controllable smart edge network.

OpenFlow is a representative south-bound protocol [1]. This protocol requires the SDN switches supporting the protocol-independent message handing. It requires the flexibility to define the field combination needed to match and the flexibility to define the operation for the implementation of the messages. This requirement requires that the forwarding chip provides flexible programmable interfaces. Nevertheless, it is difficult to realize such flexibility for merchant silicon [1]. The cost of SDN switches remains high which may hinder a wide deployment of SDN networks. Till now, they have only been used for places such as data centers that are not sensitive to cost [2]. The emergence of Xilinx Zynq series of chips provides a cost-effective but all programmable approach to implement SDN switches in a software/hardware co-design. Previous studies have shown that it is possible to achieve a good design balance between the speed of hardware and the flexibility of software [3]. As a fast-developing technology, the cost of SDN switches could be dropping quickly low enough for wide applications in smart home. So we can combine SDN technology with smart home and so on [4]. We can connect all devices of our home to a SDN network through WiFi and control this network through SDN switches [5].

The rest of this paper is organized as follows: In Section II, we present the hardware/software co-design method. We report the technical details of our SDN switch in hardware/software co-design as a case study in Section III. The evaluation results are reported in Section IV. In Section V, we summarize the

This work was supported in part by the National Natural Science Foundation of China (No. 61370231)

related work on constructing SDN devices and testbeds using embedded systems. Finally, we conclude this paper and outline some future work in Section VI.

II. Software/Hardware Co-design

A. Design Method

The Zynq-series chips combine with processors and FPGA separately, called PS (Processing System) and PL (Programmble Logic). PL has ten of thousands of programmble logic cell. PS is two powerful ARM Contex-A9 processing core. These two parts are connected by the AXI bus. PS is the leading system of the whole system and it can control PL part through the AXI bus [6]. PL reflects to PS part through address. This structure is different from the pure FPGA and the traditional SoC. We need to find a suitable method to develop the entire system.

The developing can be divided into hardware part and software part. Xilinx provides a series of the tool chain to complete the whole process. We use Vivado to finish the hardware development and SDK to finish the software part.

The hardware development includes using verilog to design custom digital logic circuits and set up the processing cores. The best way is encoding this digital logic into IP(Intellectual Property) cores and adding them into your project. Except for your custom IP cores. Xilinx and other third party organizations provide many ready-made IP cores to speed up the pace of development. Setting up the processing cores includes selecting clock source, Peripheral interface configuration and so on. Xilinx packages two processing cores into one IP core. And we can adjust parameters and connect it to other IP core in Vivado. After all IP cores connected, we need add the clock constrains and the pin constrains. Then, the synthesis and the implementation of the project will be complete by Vivado. The bitstream file describing the hardware project will be generated [7].

We design software based on hardware. If the software is simple and we can develop the program directly on bare metal. Or else we transplant a embedded operating system such as Linux on the hardware platform and the develop the software based on the OS. The detailed procedure can be found on the wiki of Xilinx. A picture provided by Xilinx describe the entire project development process as shown in Fig. 1.

B. TCL Scripting

We use TCL scripts to control Vivado to complete the development of hardware part. In the past, TCL was often used in industry rather than academia. We use it in our program. TCL is the abbreviation of the tool command language. TCL is an interpreter language with variables, procedures and control structures that can be used as interfaces for various design tools and design data. Like most EDA software supplier, Xilinx make TCL as the development language of the Vivado tool chain. TCL can provide commands to read and write local file system, support engineers to dynamically create the project directory, start building FPGA projects, in the project to add files. After circuit design, TCL can control Vivado to complete

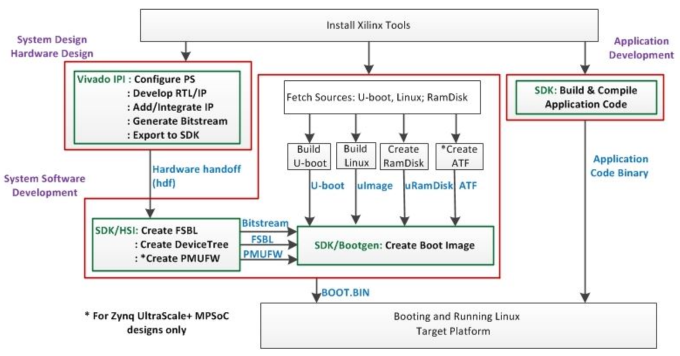


Fig. 1. The Zynq project development process.

the synthesis, implementation and simulation and bitstream generation of the design. In addition, we also use TCL to constrain time delay and circuit pin connection.

In fact, Vivado has a good graphical interface support. Why we use TCL script to complete the hardware design? First, a complete Vivado project is huge and need suitable Vivado version to open it. TCL script is only a text file and can be edit by any text editors. Second, We can use github or other software for version control and multi-person collaborative design. When the design is finished, we run the TCL script in the Vivado and can get a bitstream file described the hardware stucture. For detail information, you can refer Vivado Design Suite: TCL Command Reference Guide (Official Number:UG835) [8] and Vivado Design Suite User Guide: Using TCL Scripting (Offical NumberUG894) [9].

III. Constructing an SDN Switch

In this section, we discuss how to construct this SDN switch in a hardware/software co-design. The whole project includes hardware and software. We complete the design on the ONetSwitch development board, which used Zynq as its processing core. We separately implement switches in two ways. The hardware structure and software are both different but the design method is the similar.

A. Hardware implementation

The structure of hardware platform is unchangable. The main function of the platform is network. So it has five RJ45 interface. They are connected to dedicated commercial chips.The commercial chip completes the physical layer function. It can perform interface protocols for the physical layer of 1000BASE-T, 100BASE-TX and 10BASE-T on the cabling of Category 5 unshielded twisted pair. It means theoretical transmission speed can reach to 1Gbps. One RJ45 interface is connected to the PS part of ZYNQ through one commercial chip and the other four RJ45 interfaces are connected to PL part of ZYNQ through the other commercial chip. The detailed information of this platform can be found on www.github.com/MeshSr. Of course, the key part of hardware platform is processing core. You can refer related books for information [6].

In fact, the hardware can be divided into two parts. One is the circuit on the developing platform. It is unchangable. The other is our own custom digital logic in FPGA. Maybe calling it firmware is more accurate. We will implement a SDN switch. The SDN switch works in the second layer of the network seven-layer model. The dedicated chip on the hardware platform realizes the function of physical layer. So we need to realize the function of MAC. We can integrate the IP core modules provided by the third parties [10]. The IP core connect with physical layer chip through RGMII or GMII interface. Fig. 2 from the datasheet of the Ethernet IP core shows the structure of this IP core clearly.

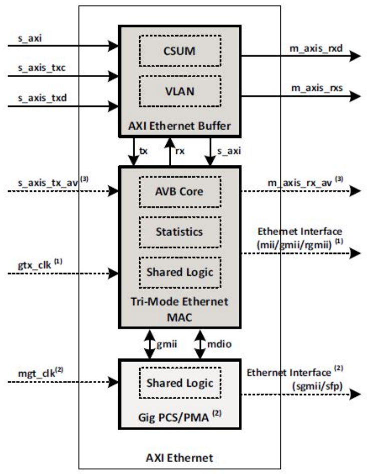


Fig. 2. The structure of the Ethernet IP core.

Only one IP core cannot work. It need to connect with clock source. And it need pass data through AXI-Streamer Interconnector and processor visit register of IP core through AXI-LITE Interconnector to control it [11]. We can collect network data through these circuit. But how can we forward network package to the right place or right port. For switches based software flow table. we just need to pass the data to the RAM through DMA then software will process them. Then they will be forwarded to the corresponding network port according the flow tables. For switches based on hardware flow table, the network packages will be passed to our custom IP core. In this digital circuit, the data will be processed and match with the flow tables. Then they will be sent to the right ports. The software will not process data. It only control the whole procedure. 3 shows the structure of the hardware.

B. Software implementation

Because SDN switch need to exchange data with peripheral and storage. If we run the software on bare computer. The software will be very complex. So we first transplant Operating system on the platform then we install our switch software on the OS. The first thing we need to do is to transplant an embedded system on the platform.

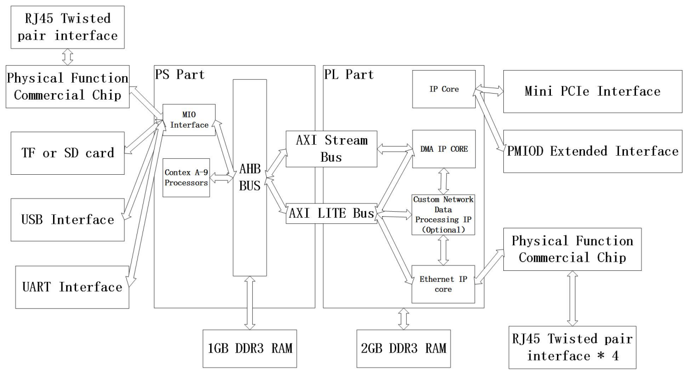


Fig. 3. The hardware structure of the SDN switch.

Transplanting operating system includes three part including boot files, kernel image and file system. The embedded system don't have BIOS. So after power up, The code that is solidified in Boot ROM runs first. This code will initialize the CPU and SD card peripherals, so that the CPU can read the code above the storage medium. Then the other peripherals will be initialized. A file named FSBL (First Stage Boot Loader) will complete this process. We could get this file through SDK. We import hardware project to SDK then we can get this file automatically. But for this project, we use a dedicated chip as an important clock source. So we need to add code to FSBL to initialize this chip. Then, we need to set up FPGA to realize the digital logic we designed before. The bitstream file can complete this work. Bitstream file is generated by Vivado. After the hardware is initialized. We need to start the operating system. A software named Uboot can support the guidance of many operating systems for many architectures. Uboot is provided by third party. We need not change anything of this software. FSBL, bit file and Uboot generate into boot file.

For an embedded system, just boot file is not enough. We need a file called device tree to manage all peripherals and Interrupt source. This file can also be generated by SDK. A whole system also include kernel and file system. Xilinx may provide official Linux kernels for specific Zynq boards. But we need to do some minor changes to support using one chip to manage four RJ45 Interface [12]. Then we need to recompile the kernel source to get the kernel image. After that, we just need to download a file system on the Internet.

In this project, we start up the switch through SD card. So we need to put these files in a SD card. We need to divide the SD card into fat and ext two partitions. Then we put boot file, device tree and kernel to the fat partition and file system to the ext partition. For the switches based on hardware flow table and software flow table, the hardware structure is different, so the bitstream file is different.

IV. Performance Evaluation

In the last section, we discuss the implementation of Software Defined Network Tested. We implemented the Network tested based both on software flow table and hardware flow table. In older to testify feasibility of switch and evaluate the performance differences between the software flow table and the hardware flow table, we construct a small network. We used RYU, which supported OpenFlow 1.3 Protocol to send the flow table. We tested our switch qualitatively and quantitatively.

A. Video delivery prototype

First, we used three computers and one switch to deploy a simplest network as shown in Fig. 4.

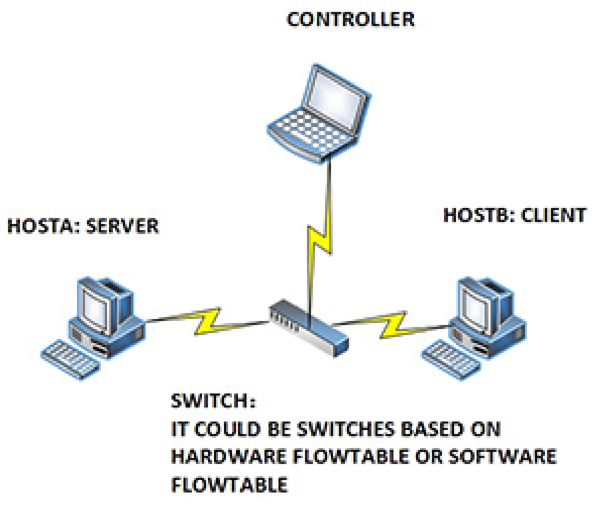
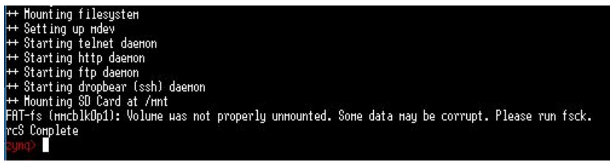


Fig. 4. Network Topology

Host A and Host B was two computers operating Linux. Switch was network tested we implemented before. Controller is a computer installing RYU controller software. We connected the hosts and controller to switch via twisted pair [13]. Then we could tested our switch.

But the switch could not work when power was on. When we implemented our switch as before, we just constructed the hardware and firmware configuration and transplanted the operating system and switch software. We needed to start the network port and configure the switch software then the switch could connect to both hosts and controller.

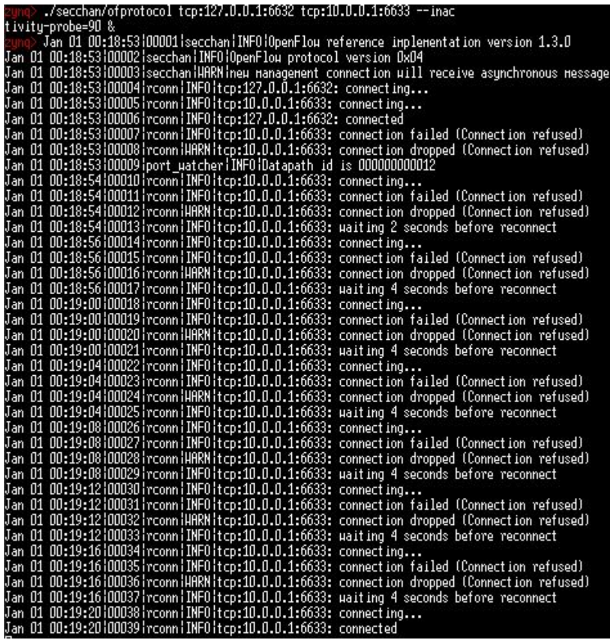
We connected a computer to our switch via serial port. Then we could interact with the Linux OS operating on our switch. We installed SD card prepared before on the switch and started the switch. We could see the initial information on hyper terminal. As 5 shown, zynq> appeared then the OS starting succeed.



```
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
++ Mounting SD Card at /mnt
FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
rcS Complete
zynq> █
```

Fig. 5. Starting Operating System

Then we opened the network port and configured the MAC addresses and the IP addresses. Then we should set default gateway to controller. Then we needed to start the switch software. We needed to configure the data path between eth1 and eth4 which separately connected to the Host A and Host B. Finally, we needed to create a secure channel connecting the switch with controller. As shown in Fig. 6, the configuration was done.



```
zynq> ./secchan/ofprotocol tcp:127.0.0.1:6632 tcp:10.0.0.1:6633 --inac
tivity-probe=90 &
zynq> Jan 01 00:18:53|00001|secchan|INFO|OpenFlow reference implementation version 1.3.0
Jan 01 00:18:53|00002|secchan|INFO|OpenFlow protocol version 0x04
Jan 01 00:18:53|00003|secchan|WARN|new management connection will receive asynchronous message
Jan 01 00:18:53|00004|rconn|INFO|tcp:127.0.0.1:6632: connecting...
Jan 01 00:18:53|00005|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:18:53|00006|rconn|INFO|tcp:127.0.0.1:6632: connected
Jan 01 00:18:53|00007|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:18:53|00008|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:18:53|00009|port_watcher|INFO|Datapath id is 000000000012
Jan 01 00:18:54|00010|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:18:54|00011|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:18:54|00012|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:18:54|00013|rconn|INFO|tcp:10.0.0.1:6633: waiting 2 seconds before reconnect
Jan 01 00:18:56|00014|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:18:56|00015|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:18:56|00016|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:18:56|00017|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:00|00018|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:00|00019|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:19:00|00020|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:19:00|00021|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:04|00022|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:04|00023|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:19:04|00024|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:19:04|00025|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:08|00026|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:08|00027|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:19:08|00028|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:19:08|00029|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:12|00030|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:12|00031|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:19:12|00032|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:19:12|00033|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:16|00034|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:16|00035|rconn|INFO|tcp:10.0.0.1:6633: connection failed (Connection refused)
Jan 01 00:19:16|00036|rconn|WARN|tcp:10.0.0.1:6633: connection dropped (Connection refused)
Jan 01 00:19:16|00037|rconn|INFO|tcp:10.0.0.1:6633: waiting 4 seconds before reconnect
Jan 01 00:19:20|00038|rconn|INFO|tcp:10.0.0.1:6633: connecting...
Jan 01 00:19:20|00039|rconn|INFO|tcp:10.0.0.1:6633: connected
```

Fig. 6. The SDN switch starts successfully.

1) Single-hop transmission: After the configuration of switch, we needed to verify the feasibility of the switch. We installed a video software on both two hosts. Host A generated video stream and transmitted to the network. Host B received the stream and play video [14]. The actual result was shown like 7.

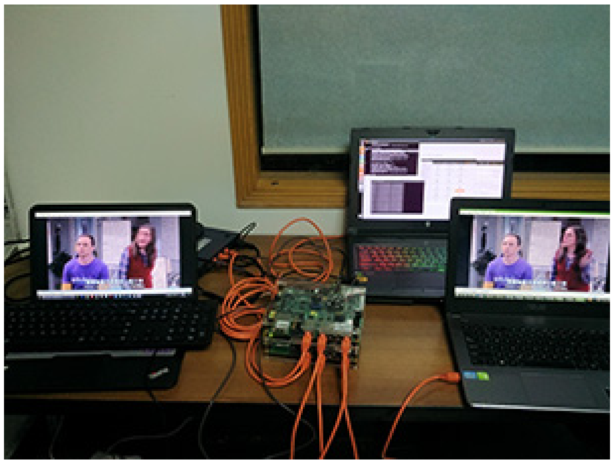


Fig. 7. The SDN switch starts successfully.

The video could be transmitted smoothly. And there was a normal time delay between server and client.

2) Multi-path transmission: After that, we construct another test network as shown in 8. The controller run the example switch program. This program can choose the complete route to pass the network data. We use server to generate video stream and transmit to the network and client receives the stream. When the video transmit smoothly. We disconnect one line as 9 shown. The video playing on client will be paused and then continue. During the paused time, the programming running in controller will choose a new path. So the video can be transferred through another path.

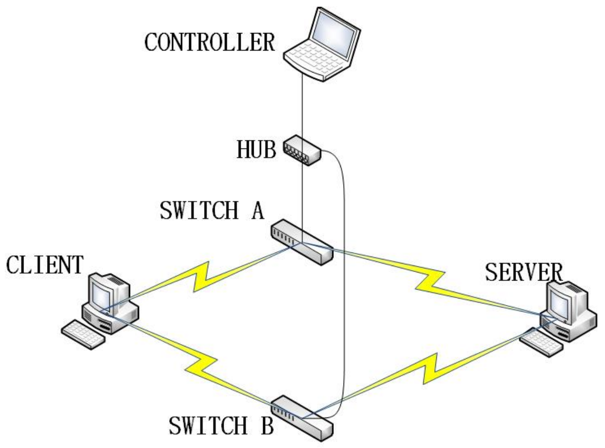


Fig. 8. A multi-path test network.

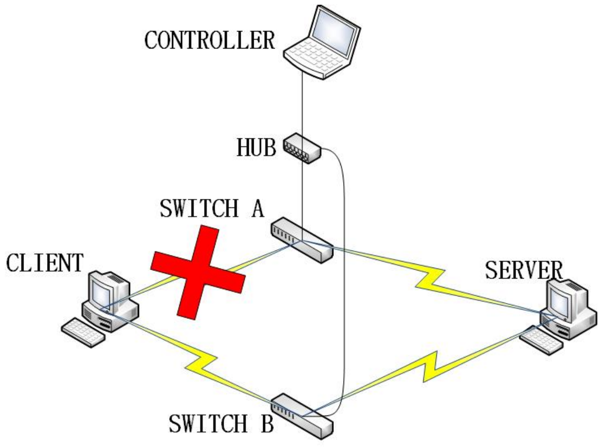


Fig. 9. One link disconnected

B. Data forwarding

After the feasibility verification of tested. We needed to do a more accurate evaluation to our switch. We implemented the switch on ONetSwitch20, ONetSwitch30 and ONetSwitch45 three different performance hardware platform. We needed to know the what difference between the switches implemented on different hardware platform and what difference between the switches implemented on the same hardware platform but based on hardware flow table and software flow table.

We just used the same topology network with 4.1. We installed iperf v2.0.5 both on two hosts.

Through the evaluation, we mainly want to achieve three purposes. First, we could obtain maximum throughput of the switch by testing the receiving rate and packet loss rate as the sending rate of sending side increasing in UDP testing mode of iperf. Second, We could get the time delay jitter at different transmission rate of different switch. Finally, we could get the time delay by ping test. We could make a contrast of the data collecting from different switch and come to a conclusion.

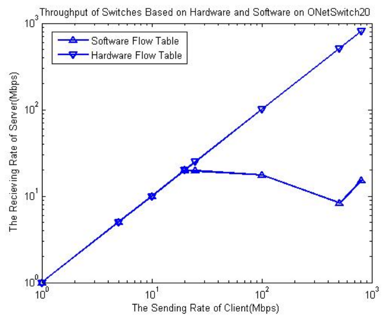
1) Maximum Throughput: Figure 10, and 11 are testing result of switches implementing on ONetSwitch20, ONetSwitch30 and ONetSwitch45. We can find that the result is similar no matter which hardware platform is implemented. But there's a big difference between hardware flow table and software flow table.

For switches based on software flow table, the receiving rate will not increase when the sending rate reach to 20Mbps. The package loss rate will also increase quickly and reach to one hundred percent. We can find the point where the package loss rate begin to increase quickly. The sending rate where the point lies is also 20Mbps. Then we can get a conclusion that the maximum throughput of the switch based on software flow table is about 20Mbps. Because performance of switch based on software flow table is determined by processor performance and three hardware platform we used use the same processor core.

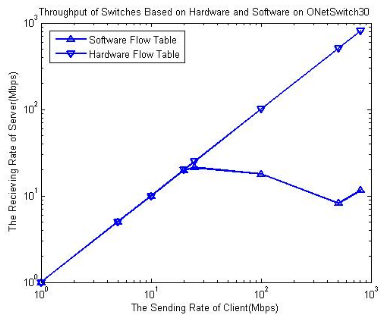
Performance of switch based on hardware flow table is completely different. The receiving rate is always increasing as the sending rate is being added. And the package lost rate is always keeping zero. We cannot observe the package lost rate increases rapidly. Because the maximum sending rate of our hosts is only 800Mbps and the switch based on hardware flow table do not reach its limit when the sending rate is 800Mbps. So we just know the maximum throughput is more than 800Mbps but we cannot get the precise throughput.

When we compare the maximum throughput of the switches based on software flow table and switches based on hardware flow table. We can find that performance is significantly improved when we use FPGA part rather than processing core to complete the network package matching and processing. As we mentioned before, From the speed of processing data, the hardware has an unparalleled advantage. And the result of experiment also validate this. The limit bandwidth of network interface we used on our switches is 1Gbps. The final result of switches based on hardware flow table is closed to the expected target. But the real performance of switches based on software flow table is far from the intended target. But we have to know although hardware is faster than software. The software has greater flexibility. How do we trade off between pure hardware and pure software and balance between speed and flexibility.

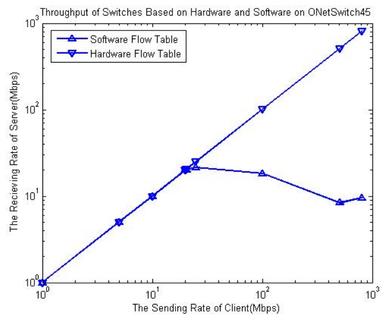
2) Delay Jitter: For quality of service of switches, there's a very important indicator called time delay jitter. The delay jitter is the absolute value of the difference between the two delays. If the network is stable, the time delay of each data



(a) ONetSwitch20

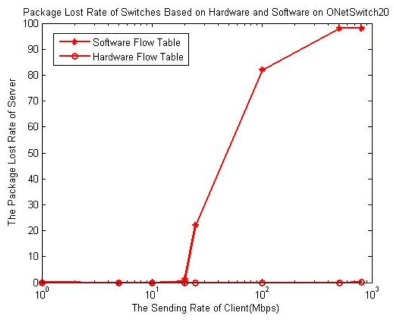


(b) ONetSwitch30

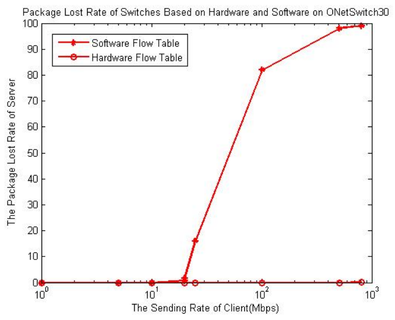


(c) ONetSwitch45

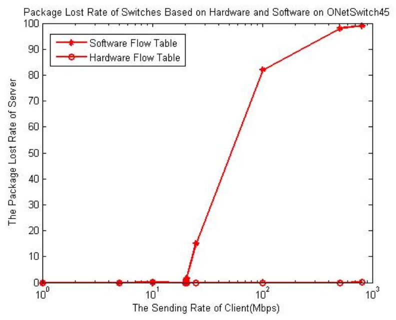
Fig. 10. Throughput comparison



(a) ONetSwitch20



(b) ONetSwitch30



(c) ONetSwitch45

Fig. 11. Packet loss comparison.

passing is almost the same. So, the user experience will be very good. Otherwise, user experience will be terrible.

We measure time delay jitter using iperf. The method of measuring is the same with previous section. Taking ONetSwitch30 as an example, we separately measure time delay jitter of switches based on hardware flow table and software flow table. The results are shown in Fig. 12.

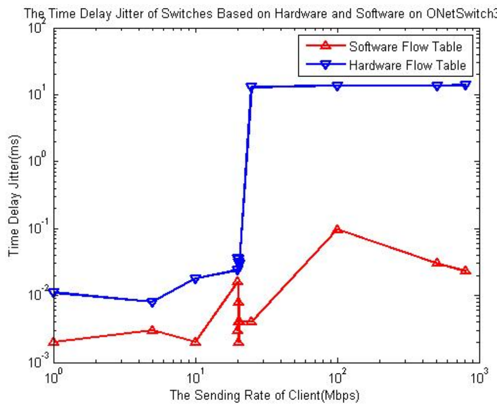


Fig. 12. The delay jitter of the switches based on hardware and software flow table

As the increasing of sending rate, the time delay jitter will increase whichever flow table is based. But time delay jitter of switch based on hardware flow table is always less than the one based on hardware flow table when the sending rate is the same. When the sending rate is higher than 20Mbps, the time delay jitter of switch based on software flow table will increase rapidly. We analyze the reason in previous section that the limitation of switch based software flow table is 20Mbps. So from the aspect of quality of service, the switch based on hardware flow table can provide more stable service.

3) Average Delay: Through the ping command, we can measure the current state of the network delay characteristics. We separately test the delay of switches based on hardware flow table and software flow table on ONetSwitch30 platform. As shown in Fig. 13, The time delay of switch base on hardware flow table is shorter than the one based on software flow table. Because there is only one switch in our test network. So it can prove the processing rate of switches based on hardware flow table is faster than switches based on software flow table.

V. Related Work

SDN technology is both useful of teaching and research. For teaching, Conventional networking courses treat today's protocols and mechanisms as fixed artifacts, rather than as part of a continually evolving system [15]. But the internet is developing and we need our student having critical thinking of existing network architecture. But the question is how

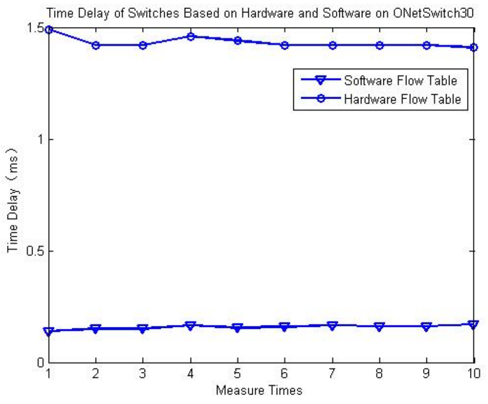


Fig. 13. The average delay of the switches based on hardware and software flow table.

to structure a course without the familiar scaffolding of the network protocol stack. But we need a way to show these abstract protocols. And researchers need to design and realize their new idea to optimize the network. But the traditional network protocols are too complicated. there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment [16]. SDN solves the problems for students and researchers. For teaching, we can present these abstract network protocols in a real, practical context and equip students to go beyond high-level architectural ideas to specific designs and implementations. For researchers, OpenFlow is a way for researchers to run experimental protocols in the networks they use every day.

In an SDN, the control plane is separated from the forwarding plane. Typically, the network OS observes and controls the entire network state from a central vantage point, hosting features such as routing protocols, access control, network virtualization, energy management, and new prototype features.The network OS controls the forwarding plane via a narrow, vendor-agnostic interface, such as OpenFlow [17]. So the forward plane is all the same when they all use the same interface to the controll plane. But the key thing is special-purpose testbeds are expensive and beyond the reach of most researchers and students. So we often use software such as mininet and so on to simulate network switches. So we can build a rapid network in one laptop. But simulation is not real enough and we finally need to deploy network in the real switches. We describe the implementation of an OpenFlow Switch on the NetFPGA platform [18]. Although this platform is all programmed. But it need a computer to support and it can work. So the appearing of ZYNQ series chips provides a new choice to solve the SDN switches. We can design hardware structure in the FPGA part and design software running in the processors. The ONetSwitch [19] [20] series developing platform provides five internet interfaces. We can design and implement our own SDN switch on it. The most critical is that the cost of this platform is significantly reduced compared with the special-purpose testbeds.

So we continuously improve our experiment testbed implemented on this platform. As we mentioned before, we use Vivado and SDK to construct the whole system. But Xilinx provides new tool named petalinux to transplant Linux to embedded system. When we finished the design of hardware circuit, we can use this tool to generate boot file, Linux kernel, devicetree and file system automatically. So we can finished our design more quickly and update in a short time. So we use new tools to construct this switches in this platform.

In addition, although we have five RJ45 interfaces on each switch. It is not enough to deploy a massive network. And many devices can only connected to network through WiFi. So we need to add wireless channel on this platform. The preliminary thought is connecting a WiFi module to the hardware platform through PCIe or USB interface and drived by operating system.

VI. Conclusion

In this paper, we applied the hardware/software co-design method, utilized the Zynq-based ONetSwitch board as the hardware platform, and reconstruct the Vivado switch hardware project using TCL scripts. We implemented the SDN switches based on software flow table using toolchain provided by Xilinx. Then, we implement the function of the flow table matching and forwarding to in FPGA so the software and hardware work together in the switches based on the hardware flow table. We conducted a series of performance evaluation experiments on an instrumented testbed. This testbed consists of various switches and hosts. Our experiments demonstrate the feasibility of constructing the SDN switches on Zynq-based embedded system in a hardware/software co-design. We reported the evaluation results of the maximum throughput, the delay jitter and the average delay of the switches based on hardware flow table and software flow table. The result comparison demonstrates the advantages of the software and hardware co-design. In the future, we plan to add the support of wireless communication on the Zynq-based switches and enhance the switches with additional features.

Acknowledgments

The authors thank Yuteng Deng and Jian Liang for their contributions in the early stage of this research. This work was supported in part by the National Natural Science Foundation of China (No. 61370231), and in part by the Fundamental Research Funds for the Central Universities (No. HUST:2016YXMS303).

References

[1] N. D. Thomas and K. Gray, *SDN: Software Defined Networks*. people's posts & telecommunications publishing house, 2014.

[2] A. Kalyaev, "FPGA-based approach for organization of SDN switch," in *9th International Conference on Application of Information and Communication Technologies*, 2015.

[3] P. R. Schaumont, "A practical introduction to hardware/software codesign," 2010.

[4] T. Zahid, D. F. Yousuf, X. Hei, and W. Cheng, "An empirical study of the design space of smart home routers," in *14th International Conference On Smart homes and health telematics (ICOST)*, 2016.

[5] T. Zahid, X. Hei, and W. Cheng, “Understanding the design space of a software defined WiFi network testbed,” in *14th International Conference on Frontiers of Information Technology (FIT)*, 2016.

[6] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media, 2014.

[7] *UltraFast Design Methodology Guide for the Vivado Design Suite*, Xilinx Inc., 2015.

[8] *Vivado Design Suite TCL Command Reference Guide*, Xilinx Inc., 2016.

[9] *Vivado Design Suite User Guide Using TCL Scripting*, Xilinx Inc., 2015.

[10] *LogiCORE IP AXI DMA Product Guide*, Xilinx Inc., 2015.

[11] *LogiCORE IP AXI Ethernet Product Guide*, Xilinx Inc., 2014.

[12] J. Corbet, A. Rubini, and G. Kroah-Hartman, “Linux device drivers, 3rd edition,” Ph.D. dissertation, O’Reilly Media, Inc., 2005.

[13] B. Lantz and B. H. nad Nick McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[14] Y. Ji, “Design and implementation of a sdn controller for data center networks,” Master’s thesis, Huazhong University of Science and Technology, Wu Han, 2016.

[15] N. Feamster and J. Rexford, “Getting students’ hands dirty with clean-slate networking,” *SIGCOMM Computer Communication Review*, vol. 41, no. 4, p. 531, 2011.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[17] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 19:1–19:6.

[18] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, “Implementing an OpenFlow switch on the NetFPGA platform,” in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008, pp. 1–9.

[19] C. Hu, J. Yang, H. Zhao, and J. Lu, “Design of all programmable innovation platform for software defined networking,” in *Open Networking Summit*, Santa Clara, CA, US, 2014.

[20] C. Hu, J. Yang, Z. Gong, S. Deng, and H. Zhao, “DesktopDC: Setting all programmable data center networking testbed on desk,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 593–594.