Building a Flexible and Scalable Virtual Hardware Data Plane

Junjie Liu[1,2], Yingke Xie[1], Gaogang Xie[1], Layong Luo[1,2], Fuxing Zhang[1,2],
Xiaolong Wu[1,2], Qingsong Ning[1,2], and Hongtao Guan[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{liujunjie, ykxie, xie, luolayong, zhangfuxing, wuxiaolong,
ningqingsong,guanhongtao}@ict.ac.cn
[2] Graduate University of Chinese Academy of Sciences, Beijing, China

Abstract. Network virtualization which enables the coexistence of multiple networks in shared infrastructure adds extra requirements on data plane of router. Software based virtual data plane is inferior in performance, whereas, hardware based virtual data plane is hard to achieve flexibility and scalability. In this paper, using FPGA (Field Program Gate Array) and TCAM (Ternary Content Addressable Memory), we design and implement a virtual hardware data plane achieving high performance, flexibility and scalability simultaneously. The data plane uses a 5-stage pipeline design. The procedure of packet processing is unified with TCAM based rule matching and action based packet processing. The hardware data plane can be easily configured to support multiple VDP (Virtual Data Plane) instances. And in each VDP instance, the pattern of packet processing can be flexibly configured. Also, it can achieve seamless migration of VDP instance between software and hardware. The hardware data plane also provides a 4-channel high-performance DMA engine which largely reduces packet acquisition overhead on software. So that software can be more involved in customized packet processing.

Keywords: Network Virtualization, Virtual Data Plane, FPGA

1 Introduction

Network virtualization has been thought as a method to experiment with innovative protocols and a feasible way to immigrate to future network [1–3]. Network virtualization virtualizes network resources, such as node, link, and topology. It enables heterogenous networks to run parallel and guarantees the isolation of each subnetwork. A router which supports network virtualization is required to fulfill various goals - scalability, flexibility, programmability, manageability, isolation and so on [5]. The router should virtualize its resources and enable multiple virtual router instances run on it with fair isolation. Each virtual router instance should be configured flexibly to process packets in their patterns. And the number of virtual router instances shouldn't be constant.

Data plane is a critical part in router architecture. It is responsible for forwarding decision, packet processing and output link scheduling [6]. It has to process millions of packets per second. For data plane in virtual router, it has to achieve virtualization, scalability, flexibility, and isolation. Those requirements can be easily achieved using sophisticated host virtualization technology [4, 10, 11, 16] (e.g. Xen). However, the performance is far from enough to satisfy practical scenario. Even after high performance I/O virtualization [21], virtual machine supervisor modification and kernel-level optimization, the performance is still inferior to raw linux performance [11].

For better performance, the data plane has been taken out from general purpose processor to platforms like NP (Network Processor), FPGA (Field Program Gate Array) or even GPU.

NP is specifically designed for network applications with pipeline of processors and parallel processing features. Nevertheless, the hardship of NP programming leads to the decline of NP usage in academic community. On the other hand, due to the popularization of NetFPGA [13], a FPGA based platform designed for network processing, FPGA is gaining popularity. Its high performance feature and the easiness of programming using HDL(Hardware Description Language) are favored by many researchers. Besides FPGA, there is a rising interest on using GPU for network processing. GPU has numerous processers and high I/O bandwidth which makes it a perfect platform for FIB (Forwarding Information Base) lookup [22]. However, the lack of platform specifically designed for network processing limits the use of GPU.

In this paper, we use FPGA to implement a virtual hardware data plane. To achieve high performance classification, we use TCAM (Ternary Content Addressable Memory) in our platform. TCAM is a special memory addressed by content. It is widely used in high-performance routers and other network devices. It stores 0, 1, or x (don't care). Each lookup of TCAM returns the address of the first entry matching the content in determined time.

The rest of the paper is organized as follows. In Section 2 we discuss some related work. By analyzing their pros and cons, we conclude our philosophies of building our data plane. In Section 3, we brief on the architecture of the hardware platform we used. And in Section 4 we show how our hardware data plane is designed and implemented. We show the overview of the design in Section 4.1, and elaborate each processing stage separately in Section 4.2. We discuss the isolation problem in Section 4.3. And in Section 4.4, we show the reference design of software for our virtual hardware data plane. In Section 5, we evaluate our hardware data plane in throughput, latency and scalability. Finally, we conclude our work in Section 6.

2 Related Work

There are already several works trying to build virtual data plane on FPGA. Anwer M. B. and Feamster [14] build eight identical virtual router instances in NetFPGA. Each instance possesses fixed forwarding table and forwarding logic. The design is preeminent in performance and isolation. But it only supports constant virtual router instances with fixed hardware resources. There is no flexibility or scalability. While their further work SwitchBlade [7] is designed in a completely different way. It pipelines and modularizes the packet processes which can be easily deployed customized protocol on programmable hardware like FPGA. The platform enables customized process to be implemented in hardware. However, the design fails to consider scalability issue.

To explore scalability, D. Unnikrishnan et al. [8] propose a scalable virtual data plane. It implements virtual router instances both in FPGA and in software which achieves better scalability. They also propose a method of dynamic FPGA logic reconfiguration. So that virtual router instance can be migrated from software to hardware and vice versa by reconfiguring FPGA. It resolves the problem of scalability in a degree, and the performance of each virtual data plane can be changed. Nevertheless, the reconfiguration of the hardware which disables the whole hardware data plane compromises isolation. In their later work [17], the reconfiguration is done using partial reconfiguration feature of FPGA. Based on this method, the reconfiguration can be done in less time without interfere other VDP (Virtual Data Plane) instances in hardware. However the number of VDP instances implemented on hardware is still constrained.

Another work need to be mentioned is OpenFlow [18]. Although OpenFlow itself does not concerns about network virtualization in its data plane, it has a relative simple yet flexible data plane. The processing of packet can be easily defined and configured using action based processing. FlowVisor [20] tries to implement virtualization in OpenFlow with a middle layer virtualizing the control plane. However, it only virtualizes the control plane, leaving the data plane unchanged.

Based on the analysis above, we have developed our philosophies on the construction of virtual hardware data plane. Firstly, the data plane should be piplelined and modularized. The pipeline will increase the performance of the hardware data plane. While modularity will facilitate the implementation of new features into the data plane. Secondly, hardware and software cooperation should be intensified. The resource in FPGA is limited comparing to increasingly powerful multicore processor in the host. The cooperation with software will boost scalability as well as programmability of the data plane. Thirdly, a unified action-based processing should be adopted to increase flexibility and utility of resources. The isolation of FPGA resources in virtual data plane not only constrains scalability but also deteriorates flexibility. Meanwhile, action-based processing saves hardware resources and increases flexibility.

3 Hardware Design

Our hardware data plane is built based on our own NAC (Network Acquisition Card) hardware platform. Figure 1 shows the architecture of NAC hardware platform. Unlike NetFPGA platform [13] which has quite scarce resources and is designed for prototype implementation and algorithm verification, NAC platform aims for practical deployment. The platform is centered with a Xilinx Virtex-5 LX155T FPGA with sufficient logic resource. The FPGA chip integrates a 8-lane endpoint block for PCI Express 1.0a providing an unidirectional data rate up to 20Gbps. It has 4 Gigabit Ethernet SFP interfaces, so that different SFP modules can be plugged in for different types of service access. Another important component is the on-board IDT75P42100 TCAM chip with a capability of 2.304Mbit. For large-volume memory storage, a 9MB QDRII SRAM chip is added to the hardware platform. Currently, this platform has been used in enterprise network as hardware accelerator in many practical systems such as intrusion detection system.

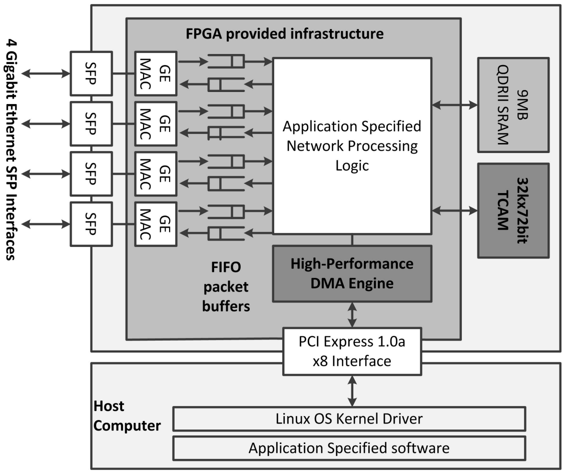

FPGA provided infrastructure
4 Gigabit Ethernet SFP Interfaces
SFP
SFP
SFP
SFP
GE MAC
GE MAC
GE MAC
GE MAC
Application Specified Network Processing Logic
9MB QDRII SRAM
32kx72bit TCAM
FIFO packet buffers
High-Performance DMA Engine
PCI Express 1.0a x8 Interface
Host Computer
Linux OS Kernel Driver
Application Specified software

Fig. 1. Architecture of NAC hardware platform

4 Data Plane Design and Implementation

4.1 Design Overview

The data plane we designed is based on the philosophies we discussed in Section 2. The design overview of the data plane is showed in Figure 2. The data plane pipeline is made up of 5 stages - VDP mapping, action matching, action processing, software processing, and forwarding.

Packets are processed as follows. Firstly, packets are gathered from the ethernet interfaces and mapped to their VIDs (Virtual data plane ID) in VDP mapping stage. Then, packets along with their VIDs are used to obtain the action result in action matching stage. The action result contains the information how the packet will be processed. Afterward, packets are processed by function modules in action processing stage according to their action results. Packets are either forwarded to the software processing stage for further process or sent to the forwarding stage for output scheduling. Software processing stage acts as an extension of hardware data plane which implements slow data plane. As the software is highly programmable, the programmability of our data plane expanded.

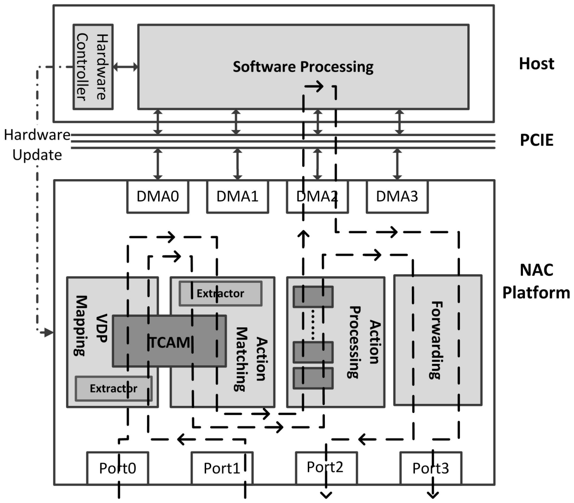


Fig. 2. Design overview of hardware data plane

In the implementation, we extracts 10-tuple packet information for rule matching in VDP mapping and action matching. The 10-tuple packet descriptor is similar to the 10-tuple flow header used in OpenFlow [18]. Hence, some innovative protocols can not be directly processed by the hardware data plane. However, we believe that in the near feature, TCP/IP stack based traditional packets will still dominate the traffic. And the minority of the traffic can be processed by software in our data plane. As the data plane is well integrated with software processing, innovation can be supported by software processing stage. What's more, the modularity of the hardware data plane enables the data plane to be reprogrammed to support new features.

4.2 Process Stages

VDP Mapping Stage. The VDP mapping stage contains two modules - information extracting and VDP mapping. Packet information is extracted from each packet for VDP mapping and used to match predefined VDP mapping rules in TCAM which returns the VID of the packet. Every VDP instance can install several mapping rules when it is created. These mapping rules are used to describe packets that belongs to the VDP instance.

The 10-tuple packet information is extracted in the implementation. It covers packet information from link layer to transport layer. Hence, VDP instance can be flexibly configured to implement different layers of virtualization [19]. For example, a layer 2 virtualized virtual data plane can be deployed using VDP mapping rules that only concerns source MAC address field, destination MAC address field and VLAN ID field. While for network layer virtualization, mapping rules should not care about fields other than source and destination IP addresses. What's more, a default VDP mapping rule is used to match all the unmatched packets to a default VDP instance which is implemented in software.

Action Matching Stage. Action matching stage also includes information extracting and action matching. The extracted fields can be different from that in the VDP mapping stage. The extracted packet information along with the VID is used to match action matching rules in TCAM. Action matching rules are made up of extracted packet fields and VID field and installed by each VDP instance. So that packets in one VDP instance wouldn't match rules in another. And within each VDP instance, rules can be flexibly configured.

In the implementation we reuse the 10-tuple packet information for action matching. Every match of action matching rule returns an index of action result using which an action result can be obtained. Action result contains information of how the packet will be processed in action processing stage. There is a default rule for each VDP instance that match all unmatched packets. Packet that fails to match explicit rules will hit the default rule and be processed according to the default action result. Usually, the default action result can be set as sending packets to the software processing stage. With more explicit rules in hardware, the VDP instance will have better performance. So the performance of VDP instance can be configured by changing the number of explicit rules in TCAM. What's more, by replacing first rule with the default rule, a VDP instance will seamless migrate from hardware to software.

Action Processing Stage and Forwarding Stage. Packets along with their action results are sent to action processing stage. In this stage, packets are processed by a pipeline of function modules according to their action results. Each function module is responsible for a specific processing function. As this stage is modularized, more modules as well as corresponding fields in action result can be added to support more patterns of packet processing in hardware.

In the implementation, an action result contains a bitmap field and various value fields. Bitmap field has several bits: port-valid bit, change-MAC bit, TTL bit, forward bit, and drop bit. While the value fields are port ID field, source and destination MAC addresses field, and DMA ID field. Port-valid bit implies the validity of port ID field through which packets will be sent. Change-MAC bit is asserted when packets should change their MAC addresses with the ones in source and destination MAC addresses field. The TTL field indicates the decrease of the TTL field in packets and recalculation of the checksum in IP header. If forward field is asserted, packet should be directly forwarded to the given output port, otherwise they will be sent to the host through the DMA channel in DMA ID field. And drop bit denotes whether those packets should be dropped or not.

The combination of those fields results in various patterns of actions. For example, host forwarding action, with the forward bit deasserted and DMA ID given, can be used as the action result of default matching rule for each VDP instance. Packets that obtain host forwarding action are sent to host through the given DMA channel.

Forwarding stage is responsible for packet output scheduling. Packets and their output port IDs from either action processing stage or software processing stage are sent to this stage. Then they are scheduled and forwarded to the their coordinating output ports.

DMA Engine and Software Processing Stage. In our work, a high-performance DMA engine is adopted in the hardware data plane. The DMA engine directly sends packet data to the reception ring buffer in host memory without interrupting CPU. While the software polls data from the ring buffer all the time and assembles the packets. One a reverse way, the software writes packet data to the transmission ring buffer in host memory and changes the buffer pointers in the hardware for notification. Then the hardware automatically requests those data from host memory and assembles packets in the hardware. In this way, the interrupt rate of system dramatic drops eliminating frequent context switching. What's more, the packets can be processed in a batched fashion further improving the performance for software processing.

Additionally, the DMA engine implements 4 independent channels for packet reception and transmission. That means there are 4 independent reception and transmission ring buffers respectively in host memory. This provides an alternative way of I/O virtualization. One of the channels can be used by supervisor for distributing packets to correspondent virtual router instances. Other three channels can be dedicated to virtual router instances which requires high-performance I/O throughput.

Packets that can not be processed in hardware, due to the limitation of TCAM entries or the absence of customized processing modules, will be sent through the DMA channel to software. The software processing stage acts as an extension of hardware data plane which adding programmability to the data plane. The high-performance DMA engine we used in this design also boosts the performance of software processing.

4.3 Resource Isolation

Virtualization requires the share of physical resources, while isolation demands the elimination of interaction. For software, the resources needed to be isolated are CPU time, allocation and access of memory, hard drive space, and so on. The interrupt due to packet reception, i.e. I/O fairness, also should be taken into consideration [12]. For work in [14], the isolation is accomplished by separation of logic resources. However, in our hardware data plane, it no longer applies.

As described above, all the VDP instances in hardware data plane share the pipeline. We think that the unified pipeline is fast enough to handle all packets. With data width of 128 bit and a synchronous clock of 200MHz, the hardware is capable of handling packets in 25.6Gbps which is far more than maximal packet rate in this platform. So packets can peacefully time-share the processing modules. The major problem is the share of TCAM. TCAM is used to store mapping rules and action rules for all VDP instances. Every instance may need several mapping rules and action rules. The management of TCAM should be easy to allocate and deallocate TCAM entries to VDP instances, and provides fair access to TCAM.

We use a slice-based TCAM management in our work. The TCAM is divided into two parts - VDP mapping rules and action matching rules. TCAM entries are partitioned into slices. The allocation and deallocation of TCAM entries is based on slices. The supervisor maintains the usage information of all slices - whether a slice is used or not and which VDP instance does a slice belong to. The supervisor also acts as an agent translating TCAM operations of virtual router instances to the physical TCAM operations. Another benefit is the potential for TCAM power consumption reduction which is critical in every scenario using TCAM. Before we send the request for a TCAM lookup, we already know which slices will be matched, either VDP mapping slices or action matching slices for a certain VDP instance. So we can only select blocks which contains those slices to perform the match instead of using the whole TCAM.

4.4 Reference Software Architecture

To facilitate the configuration of the virtual data plane, we provide two sets of API. A higher level of API can be used by control plane of virtual router instance to manage its VDP mapping rules and action matching rules logically. The lower level of API is used by the supervisor to manage TCAM allocation and perform physical TCAM entries modification.

The Pearl Project [9], aiming to build a programmable virtual router platform, uses the virtual hardware data plane described in this paper. The software architecture of the Pearl Project is showed in Figure 2. The substrate layer deals with distribution and isolation of packets as well as hardware management. LXC, a light-weight OS-level virtualization technique, is used to contain control plane and slow data plane of a virtual router instance. Three of the four DMA channels are dedicated to three virtual machines respectively, and the last one is shared by the rest virtual machines. And a VM (Virtual Machine) manager is used to manage both software virtual machines in software and VDP instances in hardware.

5 Evaluation

We evaluate our work from three perspectives - (1) throughput, (2) latency, and (3) scalability. We try to demonstrate that the hardware data plane is scalable

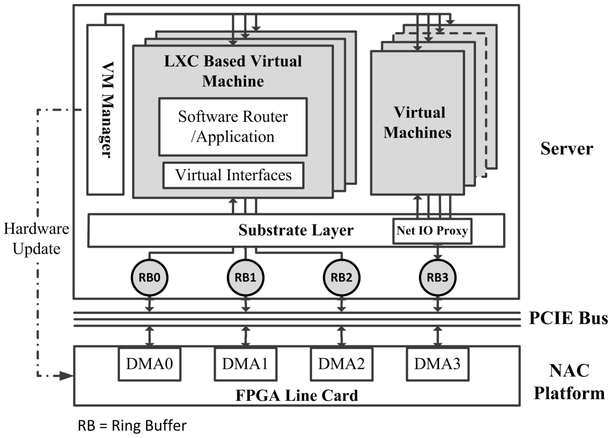


Fig. 3. Software architecture of Pearl

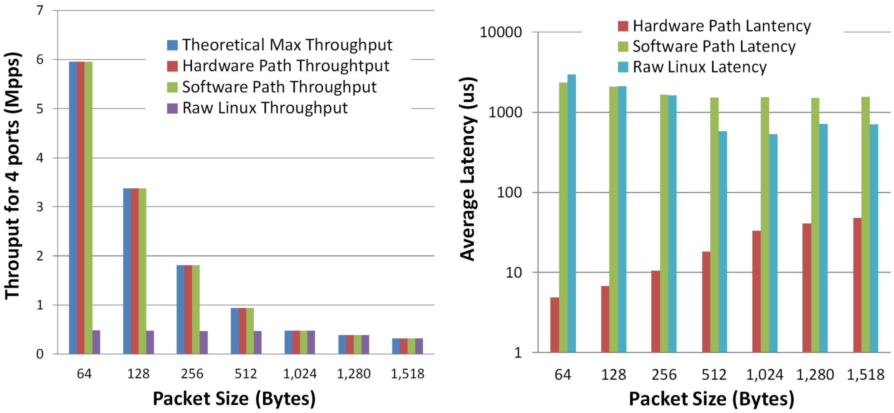
and high-performance. We use a server to host our hardware platform. It has a 3.5GHz 64-bit 8-core Intel Xeon processor, 8 GB DDR2 RAM. The OS system is a 2.6.35.9 kernel version Ubuntu. We use Spirent TestCenter [23], a powerful Ethernet testing device, to generate packets for the test.

There are two processing paths in the data plane as showed in Figure 2. One of them is hardware path - packets are directly forwarded to the output port in the hardware bypassing the software processing stage. The other is software path - packets are forwarded to the software processing stage and then sent to hardware for output.

In the evaluation of both pathes, we create four VDP instances in hardware data plane and map packets to VDP instances according to their incoming ports. While, the default action rules in these two evaluations are slightly different. In the evaluation of hardware path, packets are forwarded directly from one port to its adjacent one. While, in software path, packets are sent to software and then sent back to hardware and forwarded to adjacent ports of their income ports.

We use TestCenter to generate RFC2544 tests on throughput and latency. All four ports send packets with packet sizes ranging from 64 bytes to 1518 bytes. The testing result is showed in Figure 4.

The hardware path reaches 4Gbps line speed, so does the software path. Whereas raw Linux kernel forwarding is only capable of handling traffic in approximate 500kpps. The latency of hardware path is merely 5 milliseconds to tens of milliseconds, depending on packet size. The latency of software path comes close to raw kernel performance, about 1 to 2 microseconds. The 4 channel DMA engine is the primary contributor to the high throughput of software path. With 4 separated channels, packets can be processed concurrently so that we can benefit more from multi-core processor. What's more, as packet reception is based on ring buffer polling, there is no overhead of context switch due to interrupt. One the other hand, without interrupt, the timing for packet processing becomes uncertain which leads to the latency in software path.



(a) Throughput

(b) Latency

Fig. 4. Throughput and Latency of hardware data plane

To demonstrate the scalability of our data plane, we set up 4 to 1024 VDP instances in hardware data plane and evaluate the overall throughput. Figure 5 shows the performance of hardware data plane. We can see that the addition of VDP instance does not require additional logic resource in FPGA. Only TCAM entries are needed when VDP instance are created. So that the data plane is quite scalable. The overall performance will not be influence by the increase of VDP instances.

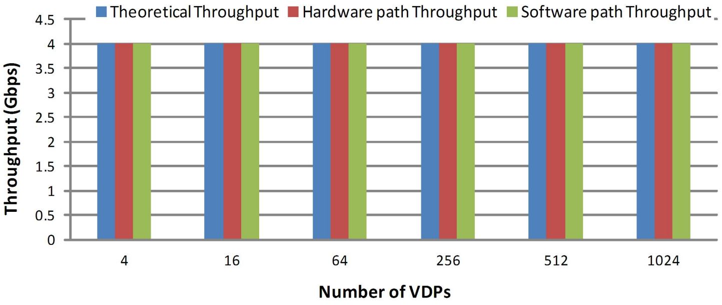


Fig. 5. Scalability of hardware data plane

6 Conclusion

In this paper, we build a virtual hardware data plane to achieve high performance as well as flexibility and scalability. The TCAM based unified pipeline design facilitates the reconfiguration of performance of VDP instances and provides flexible interface for the configuration of action based processing. The action based packet processing makes the packet processing in each VDP isntance configurable and flexible. The high-performance DMA engine promotes the performance of software and improves the programmability to support customized processing. It manages to support multiple VDP instances which means the hardware data plan is scalable. The evaluation in Section 5 shows that the hardware data plane is low-latency and high-throughput with great scalability. The increase of VDP instances in hardware would not affect the overall performance of the system.

Acknowledgement

This work is supported by National Basic Research Program of China under grant No.2012CB315801, National Natural Science Foundation of China (NSFC) under grant No. 61133015, Projects of International Cooperation and Exchanges NSFC-ANR with grant No.61061130562, Strategic Priority Research Program of the Chinese Academy of Sciences under grant No. XDA06010303, and the Instrument Developing Project of the Chinese Academy of Sciences under grant No. YZ200926.

References

1. Anderson, T., Peterson, L., Shenker, S., Turner, J.: Overcoming the Internet impasse through virtualization. Computer. 38, 34–41 (2005)
2. Niebert, N., Khayat, I.E., Baucke, S., Keller, R., Rembarz, R., Sachs, J.: Network virtualization: A viable path towards the future internet. Wireless Personal Communications. 45, 511–520 (2008)
3. Tutschku, K., Zinner, T., Nakao, A., Tran-Gia, P.: Network virtualization: Implementation steps towards the future internet. Proc. of KiVS-Kommunikation in Verteilten Systemen(KIVS09),(KiVS 2009, Kassel). (2009)
4. Rixner, S.: Network virtualization: breaking the performance barrier. Queue. 6, 36 (2008)
5. Chowdhury, N.M., Boutaba, R.: A survey of network virtualization. Computer Networks. (2009)
6. Chao, H.J., Liu, B.: High performance switches and routers. Wiley-IEEE Press (2007)
7. Anwer, M.B., Motiwala, M., Tariq, M. bin, Feamster, N.: SwitchBlade: a platform for rapid deployment of network protocols on programmable hardware. Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM. pp. 183C194. ACM, New York, NY, USA (2010)
8. Unnikrishnan, D., Vadlamani, R., Liao, Y., Dwaraki, A., Crenne, J., Gao, L., Tessier, R.: Scalable network virtualization using FPGAs. Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays. pp. 219C228 (2010)

9. Xie, G., He, P., Guan, H., Li, Z., Xie, Y., Luo, L., Zhang, J., Wang, Y., Salamatian, K.: PEARL: a programmable virtual router platform. Communications Magazine, IEEE. 49, 71–77 (2011)
10. Jiang X., Xu D.: Violin: Virtual internetworking on overlay infrastructure. Parallel and Distributed Processing and Applications. 937–946 (2004)
11. Bhatia, S., Motiwala, M., Muhlbauer, W., et.: Trellis: a platform for building flexible, fast virtual networks on commodity hardware. Proceedings of the 2008 ACM CoNEXT Conference. 72, 1–6 (2008)
12. Anwer, M.B., Nayak, A., Feamster, N., Liu, L.: Network I/O fairness in virtual machines. Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures. 73–80 (2010)
13. NetFPGA, http://netfpga.org/
14. Anwer, M.B., Feamster, N.: A Fast, Virtualized Data Plane for the NetFPGA. NetFPGA Developers Workshop, Stanford, California. pp. 90C94 (2009)
15. Liao, Y., Yin, D., Gao, L.: Network virtualization substrate with parallelized data plane. Computer Communications. 34, 1549-1558 (2011)
16. Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., Mathy, L.: Towards high performance virtual routers on commodity hardware. Proceedings of the 2008 ACM CoNEXT Conference. pp. 20:1C20:12. ACM, New York, NY, USA (2008)
17. Yin, D., Unnikrishnan, D., Liao, Y., Gao, L., Tessier, R.: Customizing virtual networks with partial FPGA reconfiguration. ACM SIGCOMM Computer Communication Review. 41, 125C132 (2011)
18. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review. 38, 69C74 (2008)
19. Chowdhury, N.M.M.., Boutaba, R.: Network virtualization: state of the art and research challenges. IEEE Communications magazine. 47, 20C26 (2009)
20. Sherwood, R., Gibb, G., Yap, K.K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G.: FlowVisor: A Network Virtualization Layer. (2009)
21. Dong, Y., Dai, J., Huang, Z., Guan, H., Tian, K., Jiang, Y.: Towards high-quality I/O virtualization. Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference. pp. 1C8 (2009)
22. Han, S., Jang, K., Park, K., Moon, S.: PacketShader: a GPU-accelerated software router. Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM. pp. 195C206. ACM, New York, NY, USA (2010)
23. Spirent TestCenter, http://www.spirent.com