PEARL: a programmable virtual router platform — **Source link** ⬀

Gaogang Xie, Peng He, Hongtao Guan, Zhenyu Li ...+5 more authors

Institutions: Chinese Academy of Sciences

Topics: Full virtualization, Core router, One-armed router, Virtualization and Packet processing

Related papers:

- Future internet architectures: design and deployment perspectives

- A survey of network virtualization

- Building Scalable Virtual Routers with Trie Braiding

- Efficient IP-address lookup with a shared forwarding table for multiple virtual routers

- Performance evaluation of open virtual routers

PEARL: a programmable virtual router platform

Gaogang Xie, Peng He, Hongtao Guan, Zhenyu Li, Yingke Xie, Layong Luo,

Jianhua Zhang, Yonggong Wang, Kavé Salamatian

▶ **To cite this version:**

PEARL: A Programmable Virtual Router Platform

Gaogang Xie, Peng He, Hongtao Guan, Zhenyu Li, Yingke Xie,
Layong Luo, Jianhua Zhang, Yonggong Wang
Institute of Computing Technology, Chinese Academy of Sciences
Kave Salamatian
University of Savoie

ABSTRACT

Programmable routers supporting virtualization are a key building block for bridging the gap between new Internet protocols and their deployment in real operational networks. This article presents the design and implementation of *PEARL*, a ProgrammablE virtuAl Router pLatform with relatively high-performance. It offers high flexibility by allowing users to control the configuration of both hardware and software data paths. The platform makes use of fast lookup in hardware and software exceptions in commodity multi-core CPUs to achieve high speed packet processing. Multiple isolated packet streams and virtualization techniques ensure isolation among virtual router instances.

INTRODUCTION

Deploying, experimenting and testing new protocols and systems over Internet have always been a major issue. While, one could use simulation tools for the evaluation of a new systems aimed toward large-scale deployment, real experimentation in experimental environment with realistic enough settings, such as real traffic workload and application mix are mandatory. Therefore easy programmable platforms that can support high-performance packet forwarding and enable parallels deployment and experiment of different research ideas are highly demanded.

Moreover, we are moving toward a future Internet architecture that seems to be polymorphic rather than monolithic, *i.e.,* the architecture will have to accommodate simultaneous coexistence of several architectures (like the Named Data Network (NDN) [1], *etc.*) including the current Internet. Therefore future Internet should be based on platforms running different architectures in virtual slices enabling independent programming and configuration of functions of each individual slice.

Current commercial routers, the most important building blocks for Internet, while attaining very high performance, offer only a very limited access to the researchers and developers to their internal component to implement and deploy innovative networking architecture. In contrast open software based routers naturally facilitate the access and adaptation of almost all their component however with a frequently low packet processing performance.

As an example, recent OpenFlow switches provides flexibility by allowing programmers to configure the 10-tuple of flow table entries, enabling the change of packet processing of a flow. OpenFlow switches are not ready for non-IP based packet flows, such as NDN. Moreover, while the switches allow a number of slices for different routing protocols through the FlowVisor, the slices are not isolated in terms of processing capacity, memory and bandwidth.

Motivated by these facts, we have designed and built a ProgrammablE virtuAl Router pLatform, *PEARL* that can guarantee high performance. The challenges are two-fold: first to manage the flexibility *vs.* performance trade-off that translates into pushing functionality to hardware for performance *vs.* programming them in software for flexibility, second to ensure isolation between virtual router instances both in hardware and software with low performance overhead.

This article describes the PEARL router's architecture, its key components and the performance results. It shows that PEARL meets the design goals of flexibility, high performance and isolation. In the next section, we describe the design goals of the PEARL platform. These goals can be taken as the main features of our designed platform. We then detail the design and implementation of the platform, including both hardware and software platform. In the next section, we evaluate the performance of a PEARL based router using the Spirent TestCenter by injecting into it both IPv4 and IPv6 traffic. Finally, we briefly summarize the related works, and conclude the paper.

DESIGN GOALS

In the past few years, several future Internet architectures and protocols at different layers have been proposed to cope with the challenges that the current Internet faces [1]. Evaluating the reliability and the performance of these proposed innovative mechanisms is mandatory before envisioning a real scale deployment. Besides theoretically evaluating the performance of a system and simulating these implementations, one needs to deploy them in a production network with real user behavior, traffic workload, resource distribution, and applications mixture. However, a major principle in experimental deployment is the "*No harm"* principle that states that normal services on a production network should not be impacted by the deployment of a new service. Moreover, no unique architecture or protocol stack will be able to support all actual and future Internet services and we might need specific packet processing for given services. Obviously, a flexible router platform with high-speed packet processing ability and support of multiple parallel and virtualized independent architectures is extremely attractive for both Internet research and operation. Based on this observation one can define isolation, flexibility, and high performance as the needed characteristics and the design goals of a router platform future Internet.

In particular, the platform should be able to cope with various types of packets including IPv4, IPv6, even non-IP and be able to apply packet routing as well as circuit switching. Various software solutions like Quagga or XORP [2] have provided such flexible platform that is able to adapt their packet-processing components as well as to customize the functionalities of their data, control and management planes. However, these approaches fail to be fast enough to be used in operational context where a wire-speed is needed. Nevertheless, by adding and configuring convenient hardware packet processing resources such as FPGA, CPU cores and memory storage one can hope to meet the performance requirements. Indeed, flexibility and high performance are in conflict in most situations. Flexibility requires more functionalities to be implemented in software to maximize the programmability. On other hand, high performance cannot be reached in software and needs custom hardware. A major challenge for PEARL is to allocate enough hardware and multi-cores in order to achieve both flexibility and high performance.

Another design goal is relative to isolation. By isolation we mean a mechanism that enables different architectures or protocols running in parallel on separate virtual router instances without impacting each other performances. In order to achieve isolation, we should provide a mechanism that will ensure that one instance can only use its allocated hardware (CPU cores and cycles, memory, resources, *etc.*) and software resources (lookup routing tables, packet queue, *etc.*) and is forbidden to access resources of other instances even when they are idle. We need also a dispatching component that will ensure that IP or non-IP packet are delivered to specified instances following custom rules defined over MAC layer parameters, protocols, flow label or packet header fields.

The PEARL offers high flexibility through the custom configurations of both hardware data path and software data path. Multiple isolated packet streams and virtualization techniques enable the isolation among virtual router instances, while the fast lookup hardware provides the capacity to achieve high performance

PLATFORM DESIGN AND IMPLEMENTATION

A. System Overview

PEARL uses commodity multi-core CPUs hardware platforms that run generic softwares as well as specialized packet-processing cards for high performance packet processing as shown in Figure 1. The virtualization environment is build using the Linux-based LXC solution. This enables multiple virtual router instances to run in parallel over a CPU core or one router instance over multiple CPU cores. Each virtual machine can be logically viewed as a separate host. The hardware platform contains a FPGA-based packet processing card with embedded TCAM and SRAM. This card enables fast packet processing and strong isolation.
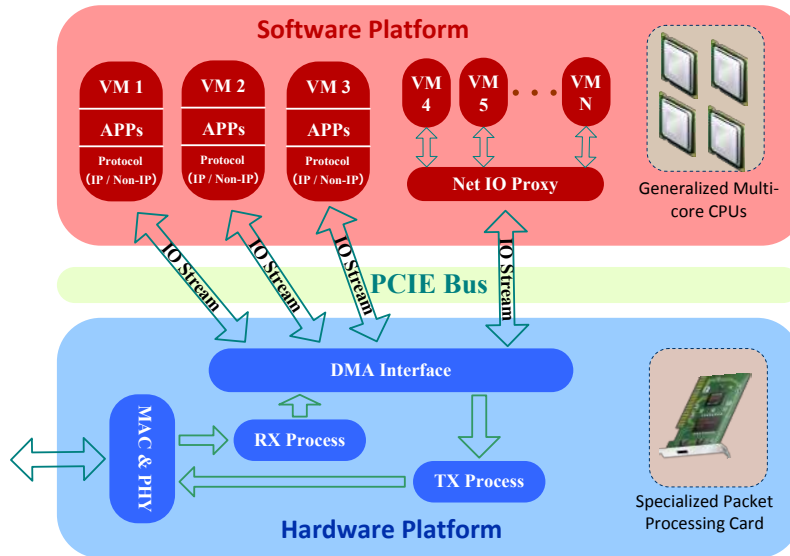

Figure 1 Overview of the PEARL architecture

Isolation. PEARL implements multiple simultaneous fast virtual data planes by allocating separate hardware resources to each virtual data plane. This facilitates strong isolation among the hardware virtual data planes. Moreover, LXC takes advantage of a group of the kernel feature (namespace, Cgroup) to ensure isolation in software between virtual router instances. A multi-stream high-performance DMA engine is also used in PEARL, which receives and transmits packets via high-speed PCI Express bus between hardware and software platform. Each IO stream can be either assigned to a dedicated virtual router or shared by several virtual routers using a net IO proxy.

Flexibility. We use TAP/TUN device as the network interface in each virtual machine. Each virtual machine could be considered as a standard Linux host containing multiple network ports. Thus, the IPv4, IPv6, OpenFlow, even Non-IP protocol stack can be easily loaded. Adding new functions to router is also convenient though programming Linux applications. For example, to load IPv4 or IPv6, Quagga routing software suite can be used as the control plane inside each Linux container.

High performance. The operations of routing table lookup and packets dispatch to different virtual machines are always the performance bottleneck. PEARL offloads these two operations into hardware to achieve high speed packet forwarding. In addition, since LXC is a light-weight virtualization technique with low-overhead, the performance is further improved.

B. Hardware Platform

To provide both high performance and strong isolation in PEARL, we design a specialized packet processing card. Figure 2 shows the architecture of hardware data plane. It's a pipeline-based architecture, which consists of two

main data paths: the transmitting and the receiving data path. The receiving data path is responsible for processing the ingress packets and the transmitting data path processes the egress packets. In what follows, the processing stages of the pipeline are detailed.

Header Extractor. For each incoming packet, one or many fields are extracted from the packet header. These fields are used for virtual router ID (VID) lookup in the next processing stage. For IP-based protocols, a 10-tuple, defined following OpenFlow [3], is extracted, while for non-IP protocols, the MAC address is extracted.
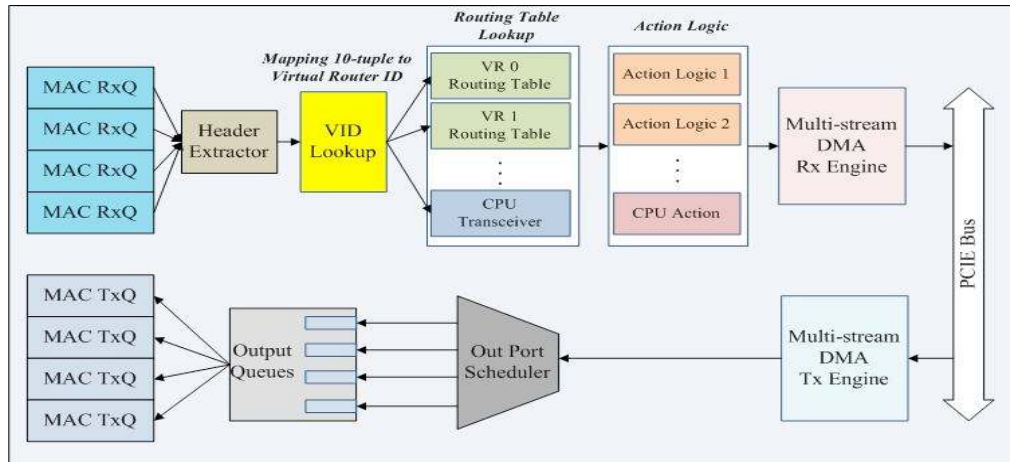

Figure 2: The architecture of PEARL's hardware data plane

VID Lookup. Each virtual router in the platform is marked by a unique VID. This stage classifies the packet based on the fields extracted in the previous stage. We use a TCAM for the storage and lookup of the fields, which can be configured by the users. Due to the special features of TCAM, each field of the rules in VID lookup table can be a wildcard. Hence, PEARL can classify packets of any kind of protocol into different virtual routers as long as they are Ethernet based, such as IPV4, IPv6 and non-IP protocols. The VID lookup table is managed by a software controller which enables users to define the fields as needed. The VID of the virtual router to which a packet belongs is appended on the packet as a custom header.

Routing Table Lookup. In a network virtualization environment, each virtual router should have a distinct routing table. Since there are no standards for non-IP protocols until now, we only consider the storage and lookup of routing tables for IP-based protocols in hardware. It is worth noting that routing tables for non-IP protocols can be accommodated through FPGA in the cards.

Given limited hardware resources, we implement four routing tables in current design. The tables are stored in TCAM as well. We take the VID combined with destination IP address as search key. The VID part of the key is performed exact matching and the IP part is performed the longest prefix matching in TCAM. Once a packet matches in the hardware, it needn't be sent to the kernel for further processing, greatly improving the packet forwarding performance.

For non-IP protocols or the IP-based protocols that are not accommodated in the hardware, we integrate a CPU transceiver module. The module is responsible for transmitting the packet to the CPU directly without looking up routing tables. Whether a packet should be transmitted by the CPU transceiver module is completely determined by software. With the flexibility offered by the CPU transceiver module, it's easy to integrate more flexible software virtual data planes into PEARL.

Action Logic. The result of routing table lookup is the next hop information, including output card number, output port number, the destination MAC address and so on, which is stored in a SRAM-based table. It defines how to process the packet, so it can be considered as an action associated with each entry of the routing tables. Based on the next hop information, this stage performs some decisions such as forwarding, dropping, broadcasting, decrementing TTL or updating MAC address.

Multi-stream DMA Engine. To accelerate the IO performance and to greatly exploit the parallel processing power of multi-core processor, we design a multi-stream high-performance DMA engine in PEARL. It can receive packets of different virtual routers from the network card to different memory regions in the host, and transmit packets in the opposite direction via high-speed PCI Express bus. From a software programmer's perspective, there are multiple independent DMA engines, and the packets of different virtual routers are directed into different memory regions, which is convenient and lockless for programming. Meanwhile, we make a tradeoff between flexibility and high performance of DMA transfer mechanism, and carefully redesign the DMA engine in FPGA. The DMA engine can transfer packets to the pre-allocated huge static buffer at contiguous memory locations. It greatly decreases the number of memory accesses required to transfer a packet to CPU. Each packet transported between the CPU and the network card is equipped with a custom header, which is used for carrying processing information to the destination, such as the matching results.

Output Scheduler. The egress packets sent back by the CPU are scheduled based on their output port number, which is a specific field in the custom header of the packet. Each physical output port is equipped with an output queue. The scheduler puts each packet in the appropriate output queue for transmitting.

C. Software Platform

Our software platform of PEARL consists of several components, as shown in Figure 3. These include **vmmd**, to provide the basic management functions for the virtual routers and the packet processing cards; **nacd**, to offer a uniform interface to the underlying processing cards outside the virtual environment; **routed**, to translate the forwarding rules generated by the kernel or user applications into a uniform format in each virtual router, and install these rules into the TCAM of the processing cards; **netio_proxy**, to transmit the packets between the physical interfaces and virtual interfaces, and **low_proxy**, to dispatch packets into low priority virtual routers which shares one pair of DMA Rx/Tx buffers. With different configuration and combination of these programs, PEARL can generate different types of the virtual routers to achieve flexibility.
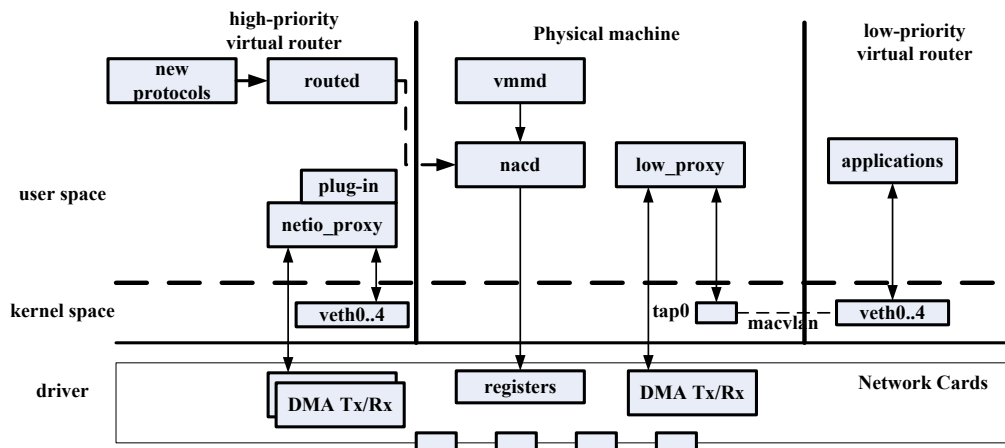

Figure 3   Packet path in the software

There are two types of virtual routers in our PEARL: the high and low priority virtual routers. Each high-priority virtual router is assigned with one pair of DMA Rx/Tx buffers, and an independent TCAM space for lookup. With the high-speed lookup based on TCAM, and efficient IO channels provided by the hardware, the virtual router can achieve the maximum throughput in PEARL platform. For low-priority virtual routers, all the virtual routers share only one pair of DMA Rx/Tx buffers and they can't utilize TCAM for lookup. The *macvlan* mechanism is adopted to dispatch packets between multiple virtual routers. The applications inside the low priority virtual routers can use the socket APIs to capture packets from the virtual interfaces. Note that each packets needs to go through at least 2 times context switch (system calls) during the transmission to the user application, resulting in a relatively low IO performance.

We take IPv4 and IPv6 as two Internet protocols to show how the virtual routers can be easily implemented on the PEARL.

High-priority IPv4 virtual router. To create a high-priority IPv4 virtual router in our platform, the **vmmd** process first starts a new Linux container with several virtual interfaces, and collects the MAC address of each virtual interface, installs these addresses in the underlying cards via the **nacd** process, so that the hardware can identify the packets heading to this virtual router, and copy the packet into the certain DMA Tx buffer which assigned to this virtual router.

Then, the **routed** process is launched in the new container. It extracts the routes through the NETLINK socket inside the virtual router and installs routes and the forwarding action in hardware, so the hardware can fill a little structure in the memory to notify the **netio_proxy** process when a packet match a route in TCAM. The **netio_proxy** process delivers the packets either to the virtual interface or directly to the physical interface according to the forwarding action in memory.

For example, most of time, normal packets will match a route in the hardware. When the **netio_proxy** receive these packets, it will directly send them through a DMA Tx buffer. An ARP request packet will not match any rules in the TCAM, the **netio_proxy** process will deliver this packet to the virtual interface, receive the corresponding ARP reply packet from the virtual interface, and then send it to the physical interface.

Low-priority virtual router. To create low priority virtual routers, a tap device is set up by the **vmmd** process (tap0). Low priority virtual routers are configured to share the network traffic of this tap device through the *macvlan* mechanism. The **low_proxy** process acts like a bridge, transmitting packets between DMA buffers and the tap device in both directions. Notes that the MAC addresses of the virtual interfaces are generated randomly, we can encode the MAC addresses to identify virtual interface where the packet comes from. For example, we can use the last byte of the MAC address to identify the virtual interfaces, if the **low_proxy** process receives a packet with source MAC address 02:00:00:00:00:00, it knows that the packet is from the first virtual interface in one of the low priority virtual routers, and transmits the packet to the first physical interface immediately. We adopted this method in the **low_proxy** process and **vmmd** process, and use the second byte to identify the different low virtual routers. It not only saves the time consumed by the inefficient MAC hash lookup to determine where the packet comes from, but also saves the space in TCAM, because all the low priority virtual routes only need one rule in TCAM (02:*:00:00:00:*).

IPv4/IPv6 virtual router with kernel forwarding. In this configuration, the **routed** process does not extract the route from the kernel routing table; instead, it enables the *ip_forward* options of the kernel. As a result, all packets will match the default route in TCAM without the forwarding action. The **netio_proxy** process transmits all these packets into the virtual interfaces, so that the kernel will forward the packet instead of the underlying hardware. The

tap/tun device is used as the virtual interface. Since the **netio_proxy** is a user space process, each packet needs two system calls to complete the whole forwarding.

User-defined virtual router. User-define packet process procedure can be implemented as a plug-in loaded by the **netio_proxy** process, which makes the PEARL extensible. We opened the basic packet APIs to the users, such as *read_packet*()*, send_packet*(). Users can write their own process module in C Language, and runs it in the independent virtual routers. For the light-weight applications which do not need to deal with huge amount of network traffic, users can also write a socket program in either high or low priority virtual routers.

EVALUATION AND DISCUSSION

We implemented PEARL prototype using a common server with our specialized network card. The common server is equipped with a Xeon 2.5GHz 64-bit CPU and 16G DDR2 RAM. The OS-level virtualization techniques Linux Containers (LXC) is used to isolate the different virtual routers (VR).

In order to demonstrate the performance and flexibility of PEARL, our implementation is evaluated in three different configurations: high performance IPv4 virtual router, kernel forwarding IPv4/IPv6 virtual router, and IPv4 forwarding in low priority virtual router.

We conducted 4 independent sub-networks with Spirent TestCenter to measure the performance of the three configurations in 1-4 VRs. Three different length of packet (64, 512 and 1518) is used (for IPv6 packet, the packet length is 78, 512 and 1518. 78 bytes is the minimal IPv6 packet length supported by TestCenter).

Figure 4 shows the throughputs of an increasing numbers of VRs using configuration 1. Each virtual data plane has been assigned with a pair of DMA RX/TX buffers, and the independent routing table space in the TCAM, resulting in an efficient IO performance and a high speed IP lookup. The result shows, when the number of the VR reaches to 2, the throughput of minimal IPv4 packet of PEARL is up to 4Gbps, which is the maximum theatrical speed of our implementation.
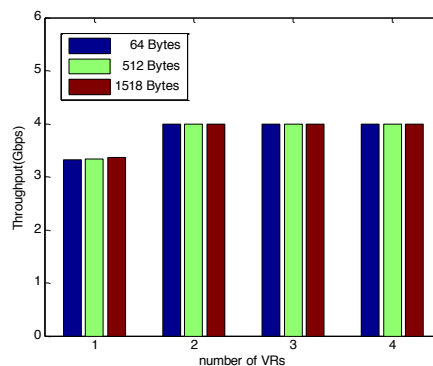

Figure 4 Throughput of high performance IPv4 virtual routers

Figure 5 illustrates the throughputs of an increasing numbers of VRs using configuration 2. Each virtual data plane has the same configuration with Configuration 1, except that there is no virtual data plane has its own routing space in TCAM. In Figure 5, the throughput of minimal IPv4/IPv6 packet forwarding is only 100Mbps when there is only one VR. It's because we used the original kernel for the packet forwarding, each packet needs to go through 2 times context switch and 3 times memory copy in our design. We can optimize this by re-implement the forwarding functions as a plug-in in the **netio_proxy** process in VR.
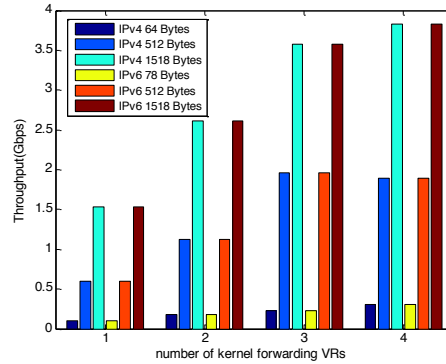
Figure 5 Throughput of kernel forwarding IPv4/IPv6 virtual routers

Figure 6 show the throughputs of an increasing numbers of low priority VRs using configuration 3. Low priority VRs shares only one pair of DMA TX/RX IO channel and can't take advantage of TCAM to speed up the IP Lookup. It can be used to verify the applications which handle little traffic (new routing protocols etc).
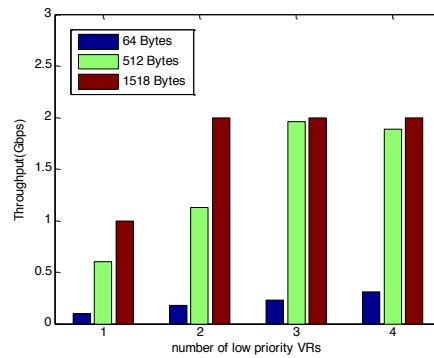

Figure 6 Throughput of low priority IPv4 virtual routers

We can see from the results, the total throughputs of the minimal IPv4 packet remains 60Mbps as the number of VR increases. The *macvlan* mechanism used for the sharing of the network traffic between multiple VRs results in a long kernel path for the packet processing, so the total performance is even lower than the IPv4 kernel forwarding in the Configuration 2. We can improve the performance by developing a new mechanism that suit for our case.

RELATED WORK

Recent researches, such as vRouter Project [4], RouteBricks [5] and PacketShader [6], have exploited the tremendous power of modern multi-core CPUs and multi-queue network interface cards (NICs) to build high-performance software routers on commodity hardware. Memory latency or IO performance becomes the bottleneck for small packets in such platforms. Due to the complexity of DMA transfer mechanism in commodity NICs, the performance of high speed PCI Express bus is not fully exploited. Meanwhile, the traditional DMA transfer and routing table lookup result in multiple memory accesses. In PEARL, we simplify the DMA transfer mechanism and redesign the DMA engine to accelerate IO performance, and offload the routing table lookup operation to the hardware platform.

OpenFlow [3] enables rapid innovation of various new protocols, and divides the function into control plane and data plane. OpenFlow provides virtualization through flowvisor, without isolation in hardware. In contrast, PEARL hosts multiple virtual data plane in hardware itself, which could offer both strong isolation and high performance.

SwitchBlade [7] builds virtualized data plane in FPGA-based hardware platform. It classified packets based on MAC address. PEARL, on the other hand, dispatches packets into different virtual routers based on the 10-tuple. With the flexibility of the 10-tuple and wildcard, PEARL has the capability to classify packets based on the protocol of any layer. Moreover, in PEARL, all packets are transmitted to the CPU for switch.

The Supercharged Planetlab Platform (SPP) [8] is a high performance, multi-application, overlay network platform. SPP utilizes the network processor (NP) to enhance the performance of traditional Planetlab [9] nodes. SPP divides the whole data plane functions into different pipeline stages on NP. Some of these pipeline stages can be replaced by custom code, resulting in an extensible packet processing. However, the flexibility and isolation of SPP is limited due to the inherent vendor-specific architecture of NP. PEARL takes advantage of the flexibility and full parallelism of FPGA to run multiple high-performance virtual data planes in parallel, while keeping good isolation in fast data path.

CoreLab [10] is a new network testbed architecture that supporting full-featured development environment. It enables excellent flexibility by offering researchers a full-virtualized environment on each node for whatever arrangement. PEARL can support the similar flexibility in software data path. Meanwhile, PEARL offers a high performance hardware data path for the performance-critical virtual network applications.

CONCLUSIONS

We aim at flexible routers to bridge the gap between new Internet protocols and the practical test, deployment. To this end, this work presents a programmable virtual router platform, *PEARL*. The platform allows users to easily implement new Internet protocols and run multiple isolated virtual data planes concurrently. A PEARL router consists of a hardware data plane and a software data plane with DMA engines for packet transmission. The hardware data plane is built on top of a FPGA based packet processing card with TCAM embedded. The card facilitates fast packet processing and IO virtualization. The software plane is built by a number of modular components and provides easy program interfaces. We have implemented and evaluated the virtual routers on the PEARL.

ACKNOWLEDGEMENT

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard (PARC) Networking Named Content, ACM CoNEXT 2009, Rome, December, 2009.

[2] M. Handley, O. Hodson, E. Kohler, "XORP: an open platform for network research," ACM SIGCOMM Computer Communication, Volume 33 Issue 1, January 2003

[3] N. McKeown, et al., "OpenFlow: Enabling innovation in campus networks," Computer Communication Review, vol. 38, pp. 69-74, Apr 2008.

[4] N. Egi, et al., "Towards high performance virtual routers on commodity hardware," presented at the Proceedings of the 2008 ACM CoNEXT Conference, Madrid, Spain, 2008.

[5] M. Dobrescu, et al., "RouteBricks: exploiting parallelism to scale software routers," presented at the Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, Big Sky, Montana, USA, 2009.

[6]   S. Han, et al., "PacketShader: a GPU-accelerated software router," ACM SIGCOMM Computer Communication Review, vol. 40, pp. 195-206, 2010.

[7]   M. Anwer, et al., "SwitchBlade: a platform for rapid deployment of network protocols on programmable hardware," ACM SIGCOMM Computer Communication Review, vol. 40, pp. 183-194, 2010.

[8]   J. Turner, et al., "Supercharging PlanetLab – A High Performance, Multi-Application, Overlay Network Platform," ACM SIGCOMM'07, August 27-31, 2007, Kyoto, Japan.

[9]   B. Chun, et al. "Planetlab: an overlay testbed for broad-coverage services," ACM SIGCOMM'03, August 25-29, 2003, Karlsruhe, Germany

[10] CoreLab. Available: http://www.corelab.jp/

BIOGRAPHIES

Gaogang Xie (xie@ict.ac.cn) received the PhD degree in computer science from Hunan University, in 2002. He is the professor at Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests include future Internet architecture, programmable virtual router platform, Internet measurement, and modeling.

Peng He (hepeng@ict.ac.cn) received his B.S. degree in Electronic and Information Engineering from the Huazhong University of Science and Technology in 2004. He is now pursuing the Ph.D. degree in Computer Architecture at the ICT, CAS. His research interests include high performance packet processing and virtual router design.

Hongtao Guan (guanhongtao@ict.ac.cn) is currently an assistant professor of ICT, CAS. His research interests include visualization technology of computer network and router. Guan studied computer science at Tsinghua University from 1999 to 2010 and obtained BE and PhD degrees. Dr. Guan had taken apart the project of BitEngine12000 during 2002 to 2005, which is the first IPv6 core router in China.

Zhenyu Li (zyli@ict.ac.cn) received the PhD degree from ICT/CAS, in 2009, where he serves as an assistant professor. His research interests include future Internet design, P2P systems, and online social networks.

Yingke Xie (ykxie@ict.ac.cn) received his Ph.D. degree from ICT, CAS, where he serves as an associate professor. His research interests include high performance packet processing, reconfigurable computing and future network architecture.

Layong Luo (luolayong@ict.ac.cn) is currently working toward his Ph.D. degree in ICT, CAS. He is currently a research assistant in the Future Internet Research Group of ICT. His research interests include programmable virtual router, reconfigurable computing (FPGA), and network virtualization.

Jianhua Zhang (zhangjianhua@ict.ac.cn) received his B.S. and M.S. from University of Science & Technology Beijing, China, in 2006 and 2009, respectively. He is currently Ph.D student at ICT, CAS. His research interest is in Future internet.

Yonggong Wang (wangyonggong@ict.ac.cn) received his B.S. and M.S. from Xidian Univiserty, Xi'an, China, in 2005 and 2008, respectively. He is currently Ph.D student at ICT, CAS. His research interest is in Future internet.

Kave Salamatian