Revealing the Necessary Conditions to Achieve 80Gbps High-Speed PC Router

Yasuhiro Ohara
NTT Communications Corporation
yasuhiro.ohara@ntt.com

Yudai Yamagishi
NTT Communications Corporation
y.yamagishi@ntt.com

Satoshi Sakai
NTT Communications Corporation
satoshi.sakai@ntt.com

Abhik Datta Banik
NTT Communications Corporation
abhik.dattabanik@ntt.com

Shin Miyakawa
NTT Communications Corporation
miyakawa@nttv6.jp

ABSTRACT

The core networking research community is witnessing a trend that many researchers attempt, and some achieve, a high speed packet processing rate using software-based approach. Yet, the extent of performance that can be obtained by deploying only Personal Computers (PCs), or Commercial Off-The-Shelf (COTS) devices, remains as an open-ended question. This is because 1) industrial achievements often involve custom hardware developments that are not counted as COTS devices, and their technical details are not revealed, and 2) research papers do not focus on the goal to answer the question of the best performance achievable using only COTS devices.

In this paper, we show our implementation of an 80Gbps PC router using only COTS devices along with our configuration details. We demonstrate achievement of 128B 80Gbps full wire-rate packet forwarding with routing lookups of 500K BGP full-route routing table, using a recent high speed routing lookup technology called Poptrie [1]. We further reveal the conditions that are necessary to construct the 80Gbps high speed PC router by presenting the performance comparisons with and without the specific parameter settings. We conclude with discussion on the promising prospects of software router application in datacenters.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Network**]: Internetworking—*Routers*; C.4 [**Performance of Systems**]: *Performance attributes*

General Terms

Performance, Design

Keywords

Intel® DPDK

1. INTRODUCTION

With the ever growing demands of extremely high speed data transfer and packet processing rate, it has become imperative to enhance the speed of the computer systems to keep pace with the 100 Gigabits Per Second (Gbps) circuit speeds. The current commodity datalink for a server in datacenter is already 10 Gigabit Ethernet (10GbE), and datacenter edge switches are commonly using 40GbE. 100GbE started to appear in the server adapter market, and is expected to become pervasive soon.

Scale-out approach, where a large number of simple inexpensive devices are used in parallel to leverage better performance, is considered a good way to enhance both the performance and the service scale of the computer systems. For the simple inexpensive device, Commercial Off The Shelf (COTS) device is suitable, because they can take the advantage of economies of scale to reach a reasonable price, making it feasible to deploy a large number of such devices in parallel. Furthermore, with the advent of Cloud Computing, Service Providers (SPs) are aggregating their computer systems in the datacenter, accelerating the scale-out approach, trying to gain more efficiency out of economies of scale. The result of this trend manifests as hundreds of thousands of COTS devices being located in a datacenter, driven by software. In such environment, SDN and NFV [2] are expected to reduce the costs (CAPEX & OPEX) and at the same time improve the system performance and the reliability (e.g., recovery time in the face of failure).

Thus, what is necessary is a high speed software that is capable of processing 100 Gbps on the COTS devices. Over the past few years, the enhanced packet processing capability of commodity hardware have paved the way for high speed software systems based on COTS devices. The architectural performance problem in processing small packets at wire speeds has been resolved by the introduction of frameworks facilitating high speed packet transfer, such as netmap [12] and Intel Data Plane Development Kit (DPDK) [7]. Our research interest is whether if we can construct the 100 Gbps capable system only using those software and the COTS devices. Specifically, we challenge the performance of *routers* using only COTS devices; i.e., software-based routers or PC routers. We further tackle the high speed large scale routing table lookup problem by adopting a recent promising technology called Poptrie [1].

Our objective in this paper is to achieve wire-rate 80Gbps forwarding performance in 128 byte-sized (128B) ethernet frames, using only COTS devices. Except otherwise explicitly noted, we refer to the achievement of the theoretical performance limitation of 40Gbit Ethernet in 128B ethernet frames as "wire-rate" in this paper. This is because that current commodity 40GbE server adapter cards seem to support the packet processing rate of the theoretical performance limitation only when the size of ethernet frame is more than 128 bytes. From the experiment in this paper, we believe that if NICs started to support the higher pps such as 64B wire-rate speed, we can easily extend our implementation to support the higher speed.

2. RELATED WORK

Click [8] by Kohler et al. is one of the pioneering works in the field of developing configurable and flexible software router. Click apportions router functions like queuing, classification of packets, scheduling, and interfacing with networking devices to independent modules called elements and thus provides flexibility to the end user to develop customized router without compromising speed or functionality. Click IP router achieves a maximum loss-free forwarding rate of 0.33M 64-byte packets per second in its initial form and with inclusion of I/O libraries of netmap, has been reported to achieve 3.95M packets per second presently [12]. Click heralded renewed interest to develop enhanced versions of software routers. RouteBricks [4], uses Click environment on COTS hardware and demonstrates routing performance upto 35Gbps by implementing maximum parallelization of routing functionality using multiple CPU cores and multiple queues on NICs. PacketShader [6], a high-performance software router platform uses Graphic Processing Units (GPUs) for packet processing. PacketShader attempts to utilize the enormous parallel processing power of GPU to address the CPU bottleneck in software routers and uses Linux as a host, by improving its network stack optimize packet I/O by incorporating enhanced packet buffer, parallel hardware processing, Non-Uniform Memory Access (NUMA) and multiple core CPU scalability. PacketShader achieves 39 Gbps for IPv4 and 38 Gbps for IPv6 with 64B packets. ClickOS [9], a Xen-based virtualized software middlebox processing platform, actualizes NFV with its capability of running hundreds of middleboxes concurrently on COTS hardware, offering processing speed of millions of packets per second and producing very low packet delays. ClickOS virtual machines are small (5MB), boot quickly (about 30 milliseconds), add little delay (45 microseconds). Experiments with ClickOS have yielded around 30Gbps packet forwarding using single low-end server. Although ClickOS based IP router produces a mediocre throughput of around 4Mpps for 128B packets, the significant feature of ClickOS is the use of Click as the principal abstraction for middleboxes and creating a customized operating system to run Click configurations because such specialization facilitates optimization of the runtime of middleboxes, enabling them to boot in milliseconds, while supporting a wide range of functionality.

Radisys [11] published the benchmark result of 128B 80Gbps throughput. It is similar to our work in that both use Intel x86 architecture, Intel DPDK, and the Intel xl710 40GbE server adapter to achieve the performance. It is different, however, that Radisys developed a custom hardware (x86 blade as their product), and their experiment did not seem to include the routing lookup process of a large routing table.

3. IMPACT OF HIGH-SPEED SOFTWARE ROUTER TECHNOLOGY

Today's commercial networks are using hardware forwarding technology based either on ASIC or FPGA to transfer the packets between switches and routers at the circuit's maximum line speed. Switches and routers deployed in the commercial backbone network are quite expensive, and the prices often go beyond 1 million USD per box. Hence, if we can use x86 architecture based COTS devices or PCs, with the 40G/100G bps Ethernet line card interfaces at the prices less than 100K USD or so to replace the hardware based forwarding machines, operators can reduce the CAPEX drastically. Especially at the edge of the network, such as in the data center for instance, since we do not need to worry about increase in RTT (round trip time) so much, we believe we can replace expensive equipments by PC-based ones to unify (or at least reduce) the hardware profiles for both the network equipment and the servers in the environment. This means that operators can reduce OPEX also.

Another advantage worth noticing is the use of open source code. The use of fully open source code based

Table 1: The list of hardware and software of the 80Gbps router.

Kind	Product Name	Cost[†]
M/B:	Supermicro X10DAX & Chassis	\$2K
CPU:	Intel Xeon E5-2687WV3 ×2	\$6K
Memory:	DDR4-2133 16GB ×16 = 256GB	\$4K
NIC:	Intel XL710-QDA1 ×4	\$4K
OS:	Ubuntu 14.04	–
Data Plane:	Intel DPDK 2.0.0	–
Lookup:	Poptrie [1]	–
	Total:	\$16K

† rounded up to the nearest 100K JPY and converted as 100K JPY = \$1K (USD).

software routers is beneficial because we can debug, extend, and modify the functions and the performance of the routers as we wish. From the operator's perspective, the coordination of routers will become easier in BSS/OSS, along with the network/cloud orchestration systems.

4. HOW TO CONSTRUCT THE 80GBPS PC ROUTER

In this section we explain how we constructed our PC router implementation. What hardware parts we purchased, the reason why, what software we installed, and how we set it up. We list here the items that will make the key differences between the achievement of 80Gbps performance and the case the performance cannot fulfill the goal.

4.1 Hardware and Software

The hardware and software we have selected for the 80Gbps router are summarized in Table 1. Supermicro's X10DAX was selected as the motherboard (M/B). We considered this as the most reasonable and high speed M/B at the time of purchase, due to the supported CPU, number of PCIe Gen3 x8 slots, the supported type and size of memories, and so on. As we will see later, apparently the memory size was not an issue; smaller memory is probably sufficient. Given the experiment results in the later section that we had more CPU/memory capacity than the NIC's bandwidth, the number of PCIe Gen3 x8 slots seems more important to achieve the higher performance. If the M/B provided more than five slots, we could test a version of the PC router with more number of 40GbE cards. They are, however, the discussions only in retrospect: we were not sure if we could achieve 80Gbps in the first place, at the time of purchase.

4.2 RSS Configuration

In order to achieve wire-rate 40Gbps speed, the use of multiqueues (Receive-Side Scaling: RSS) was necessary.

Apparently the DPDK's default setting, i.e., ETH_RSS_IP was not working properly, even when we randomized the source and destination IP address field in the generated traffic. We needed to create a new RSS configuration option in the DPDK application, so that we can specify "all" RSS hash functions. "all" RSS configuration is meant to enable as much flow classifications as possible in the NIC, and configured specifically as:

```
(ETH_RSS_IP | ETH_RSS_TCP | ETH_RSS_UDP |
ETH_RSS_SCTP | ETH_RSS_L2_PAYLOAD)
```

From the time we started to use the "all" RSS option, the 40GbE wire-rate performance (for each port) was achieved. We have not found out why the default ETH_RSS_IP setting is not working. The result of later experiment supports that the RSS was not working with the default configuration: the performance result of the default RSS configuration was very similar to the case where only one (core, queue) pair was assigned per network port.

The "NoRSSOpt" label in the later experiments indicates that the "all" RSS option was not specified, and the configuration falls to the default (non-working) setting.

4.3 Number of (CPU Core, NIC Queue) Pairs

In order to fulfill the 128B wire-rate for each 40GbE ports, only two (core,queue) pairs are necessary to achieve the performance in our configuration. We allocate and assign two (core,queue) pairs for each of the four ports, hence used eight CPU cores and NIC queues in total. We used the following setting specifically, for the (core,queue) pair assignment:

```
--config '(0,0,4),(0,1,5),(1,0,6),(1,1,7),
(2,0,10),(2,1,11),(3,0,12),(3,1,13)'
```

In the later expriments, the label "Core1" indicates that only one (core,queue) pair was specified for each port. The "Core1" configuration is as follows.

```
--config '(0,0,4),(1,0,6),(2,0,10),(3,0,12)'
```

4.4 Hugepage Memory

The use of Hugepage is mandatory in DPDK. As far as we have tested, all the memory configuration (both 2MB and 1GB hugepages) worked fine, as long as the DPDK application could be started without explicit memory configuration error.

We used 1GB hugepages basically. We specified "default_hugepagesz=1G hugepagesz=1G hugepages=8" in the kernel boot option to allocate 8 1GB hugepages.

DPDK example program, **l3fwd**, accepts the option "–socket-mem" in units of MB. For example, we sometimes used the following arguments in invokation of **l3fwd**:

```
--socket-mem 4096,4096
```

However, as we have tested, even `--socket-mem 1,1` worked without any performance degradation. Thus, we suspect that, for our performance test, allocation of only one 1GB hugepage for each CPU socket is sufficient to achieve the wire-rate performance.

Since we could not find any performance difference among memory allocation settings (other than those cases where the DPDK ceases with the explicit errors), we omit the results in this paper.

4.5 PCI Flag: Extended Tag and Max_Read_Request_Size

Extended Tag flag of PCI-e specification [10] is to increase the maximum number of outstanding requests to 256, from its default 32.

There was a significant performance difference between on and off of the Extended Tag. We will see the performance difference later in Section 5. "ExtTagOff" label in the later experiments denotes the disabled setting of Extended Tag.

Max_Read_Request_Size sets the maximum size for a read request. It is said to be influential for the performance when the forwarding packet is short and a high processing rate is required. The recommended setting is to change it to the minimum value: 128 (bytes). However, in our experiments changing the value to 4096 bytes did not make any performance difference, so we omit the experiment results from this paper. All the test results shown in this paper used the Max_Read_Request_Size of 128 bytes.

Enabling Extended Tag and setting Max_Read_Request_Size to 128 can be set collectively at once by the following instruction:

```
setpci -s <PCIeBusID> A8.W=012F
```

Additionally, Extended Tag can be set in the Intel DPDK configuration (config/common_linuxapp):

```
CONFIG_RTE_PCI_CONFIG=y
CONFIG_RTE_PCI_EXTENDED_TAG="on"
```

4.6 isolCPUs

Even though the DPDK is using the Poll Mode Driver (PMD) and conducts the busy loop to poll the NIC on the assigned CPU cores, the Operating System (OS) and its kernel can still assign other tasks to the CPU cores. If the assigned CPU core is posed with another task to process, the number of packets that was processed in the NIC port will become lower, due to the processing time for the other task.

In order to prevent the kernel from assigning some other tasks to the CPU cores assigned by the DPDK, we specified "isolcpus=4–18" in the kernel boot option. This setting will prevent the kernel from assigning some other tasks to the specified CPU cores; it will avoid the specified CPU cores and assigns the task to other CPU cores.

If there is no task to assign to the CPU core, there will be no intervention to the DPDK process, even without the isolcpus setting. It seems that our experiments were the case; we did not see any performance difference with or without the isolcpus option. However, if there are some other tasks of significant size, it will definitely affect the performance, and the specification of the isolcpus option is recommended.

Since we could not find any performance difference, we omit the experiment results in this paper. All the test results shown in this paper are with the isolcpus option set.

4.7 Intel DPDK Configuration

In the DPDK configuration (i.e., config/common_linuxapp), we had the following settings:

```
CONFIG_RTE_LIBRTE_I40E_16BYTE_RX_DESC=y
CONFIG_RTE_LIBRTE_I40E_ITR_INTERVAL=1020
```

The latter setting was found irrelevant to the performance. Hence it was changed back to its default:

```
CONFIG_RTE_LIBRTE_I40E_ITR_INTERVAL=-1
```

The results shown in Figure 2 and 3 are with the interval value of 1020. Others had the default setting.

4.8 Routing Lookup Algorithms

We have three options in the routing lookup algorithms. They are **LPM**, **LPMnoSSE**, and **poptrie**.

DPDK's **l3fwd** example program provides Longest Prefix Matching (LPM) routing lookup mode. This is based on the DIR-24-8 algorithm [5]. Furthermore, **l3fwd** provides ENABLE_MULTI_BUFFER_OPTIMIZE option that uses SSE4.1 instructions to process 4 packets at a time. We refer to the enabled and disabled versions of the SSE4.1 optimization as **LPM** and **LPMnoSSE**, respectively. **LPM** (SSE-enabled version) was the default setting of the **l3fwd** program. Note that **l3fwd** supports only 1024 routes for the total number of routes, and librte_lpm supports only 256 /24 ranges for the number of prefixes whose length is more than /24. We have modified it to support routing tables with extended number of routes, as follows.

```
-#define IPV4_L3FWD_LPM_MAX_RULES 1024
+#define IPV4_L3FWD_LPM_MAX_RULES 1048576
-#define RTE_LPM_TBL8_NUM_GROUPS  256
+#define RTE_LPM_TBL8_NUM_GROUPS  (256*256*8)
```

Furthermore, in order to test the feasibility of the recent high-speed routing lookup algorithm, we adopt Poptrie [1] in the **l3fwd**. This is refered to as **poptrie**.

When the random destination addresses are used in the benchmarking traffic, there are cases wherein the

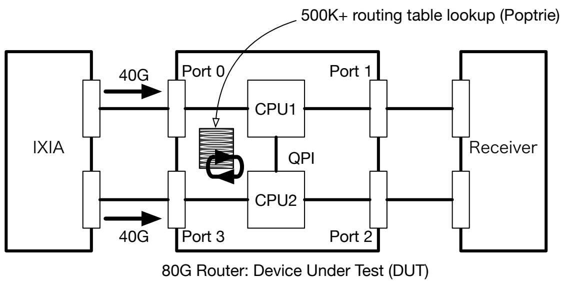
500K+ routing table lookup (Poptrie)
40G
Port 0
Port 1
CPU1
IXIA
QPI
Receiver
CPU2
40G
Port 3
Port 2
80G Router: Device Under Test (DUT)

Figure 1: The benchmark configuration.

Table 2: Basic Setting.

Level	Name	Settings
OS	Hugepages	1GB page ×8
	isolcpus	4–18
PCIe	ExtTag	On
	MaxReadReqSize	128 Bytes
DPDK	`I40E_ITR_INTERVAL`	1020 or -1‡
	RSS	–rss all
	socket-mem	4096,4096
	core assingments	2 (core,queue) pairs per port
App.	Routing lookup	Poptrie

‡ See Section 4.7.

matching route is not found. In this case, in order to still test whether the higher throughput is possible, the packets for which the nexthop port cannot be determined by the routing lookup, and should be lost in the ordinary IP forwarding process, are still forwarded to the output port. When no matching route is found, nexthop port ID (port 1 or 2) is determined based on the Least Significant Bit (LSB) of the destination IP address (which is assumed to be random).

5. EVALUATION

5.1 Network Configuration

We used IXIA (Optixia XM12 with two HSE 40/100GE TSP1-01 card) as the load generator. It is shown in the left in Figure 1.

The IXIA has two 40GbE ports, that is connected to the each of the two ports (Port 0 and Port 3) of the 80Gbps router. The other two ports (Port 1 and Port 2) are connected to the other PC, called "Receiver", shown in the right in Figure 1. The Receiver runs the DPDK-supported **pktgen**, and the **pktgen** was used to count the performance (i.e., the results in Gbps and Mpps).

Port 0 and Port 1 are under the CPU 1, and Port 3 and Port 2 are under the CPU 2. In Supermicro's X10DAX there are two QPI of 9.6GT/s between the CPU sockets. On each CPU core a routing lookup thread is run.

5.2 Routing Table

We prepared two routing tables: **linx**, and **ntt**.

As **linx**, we prepared a BGP full-route routing table from Routeviews project [3]. The routing table was extracted from peer 46 of LINX route dump at 2014/12/17 00:00 a.m. It contains 518,231 routes.

As **ntt**, we have collected a routing table from our international backbone in 2015/01/09. It contains 531,489 routes. The unique point in **ntt** is that it is the routing table actually used in the real field, with 16,140 routes that is longer than /24.

The next-hop fields of all routing tables were set to the port ID. Each next-hop port is assigned as the round-robin (toggle) fashion between the port 1 and 2: for example, 1.0.0.0/24 was destined to port 1, 1.0.4.0/24 was desgined to port 2, 1.0.5.0/24 was destined to port 1 again, and so on.

5.3 Traffic Pattern

We defined three traffic patterns: **straight**, **cross-cpus**, and **random**.

In Figure 1, if the traffic received in Port 0 is destined to Port 1, then the traffic is **straight**. On the other hand, if the traffic received in Port 0 is destined to Port 2, and the traffic received in Port 3 is destined to Port 1, then the traffic is called **cross-cpus**. We used the first two routing table entries 1.0.0.0/24 (which nexthop is Port 1) and 1.0.4.0/24 (which nexthop is Port 2) of **linx** routing table to control the **straight** and **cross-cpus** traffic. For example, if we configure IXIA to ingest the traffic to the Port 0 with the random IP destination address within the range of 1.0.4.0/24 (which is destined to Port 2), then the traffic is **cross-cpus**.

Another traffic pattern is **random**. This traffic consists of the packets with the random IP destination address from the full IP address space range. This meant to use all the entries in the routing table.

In the **straight** and **cross-cpus** traffic, the matching routing table entries are always one of the two, so it is CPU cache friendly. In contrast, in the **random** traffic case, all the routing table entries are accessed and hence it is not CPU cache friendly.

5.4 Basic Setting Summary

The setting where we succeeded to perform the 128B 80Gbps wire-rate is summarized in Table 2. Based on this setting in the table (called "basic setting"), we modified a parameter from the basic setting and checked if the performance was different. The results are shown in the next section.

5.5 Comparison by parameters

Figure 2 and Figure 3 are the comparisons by param-

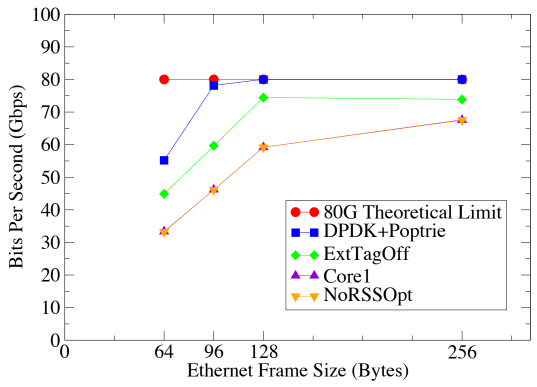

Figure 2: Straight traffic on Linx (bps).

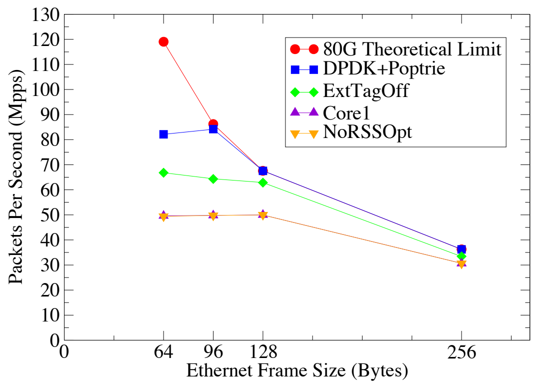

Figure 3: Straight traffic on Linx (pps).

eters, for bandwidth in Gbps and packets per second in Mpps, respectively, for the forwarding performance. The traffic used here was **straight**, and the routing table used was **linx**. Performance evaluation for each setting is conducted five times and the results are averaged. The results of Figure 2 and Figure 3 are data from the same experiments: the resulting performance figure can be calculated from one to the other.

The label "80G Theoretical Limit" denotes the theoretical maximum value of each performance. For bandwidth it is always 80Gbps regardless of the Ethernet frame size. For packets per second, it depends on the Ethernet frame size: for example, for 64B the theoretical maximum pps to fill 80 Gbps bandwidth is 119.05 Mpps, for 96B it is 86.21 Mpps, and so forth.

The label "DPDK+Poptrie" refers the performance results of the basic setting explained in Section 5.4. "ExtTagOff", "Core1", and "NoRSSOpt" are the ones that are modified in one parameter from the basic setting; they are described in Section 4.5, 4.3, and 4.2, respectively.

For the basic "DPDK+Poptrie", we can say that from 128B and above, the 80Gbps wire-rate is achieved, as shown in Figure 2. It is interesting that when we disable the Extended Tag PCIe flag, the performance decreases in 6–7 Gbps for 128B–256B.

When we allocate only one (core,queue) pair per NIC port (this is indicated by "Core1" label), the performance degradation is worse: for 128B it is 59.17 Gbps, decreasing in more then 20Gbps. Another interesting results from the figure is that "NoRSSOpt" exhibits almost the same performance result with the "Core1" label, suggesting that not specifying the RSS option makes it fail to split traffic in two queues in RSS, resulting in the same situation with "Core1" where only one (core,queue) pair is used effectively.

5.6 Cross-CPU Traffic

It was somewhat surprising to us, that there is no performance difference between traffic patterns, **cross-cpus** and **straight**. We suspect that the capacities of CPU and inter-CPU bandwidth have not reached its limitation yet, and thus, we can extend our implementation to a higher performance.

Since we did not see any performance difference, the experiment result is omitted in this paper.

5.7 LPM v.s. Poptrie

Figure 4 and 5 illustrate the bandwidth and the packet per second performance comparison between Poptrie, LPM, and LPMnoSSE, on **ntt** routing table. **linx** routing table showed the similar results, hence we omit the figure.

Since the traffic pattern is **random**, the destination IP address of the packet is random among all the IP address space. On the other hand, routing table is skewed (some routes are large, and some routes are small in size). Because we toggled the output port (1 and 2) one by one for each routes, the number of packets delivered in each port is also skewed, i.e., not equally forwarded. Since the input traffic is fully 40 Gbps, and the output traffic is skewed in port, either port is overloaded and drop some packets, and the other port lacks some packets. This is why we lose some traffic. For example, on the **random** traffic on **ntt** routing table, **poptrie** exhibits 79.198 Gbps, i.e., approximately 800 Mpps short for wire-rate. However, port 1 exhibits full 40 Gbps, and we suspect this is the overloaded port in this case.

LPM also succeeded to perform at 80 Gbps wire-rate in 128B packets. This is because that SSE optimization was effectively functioning. When we disable SSE optimization (i.e., **LPMnoSSE**), the LPM algorithm fails to perform at 80 Gbps wire-rate, as shown in the figures. As for the fair comparison, **poptrie** should be compared to **LPMnoSSE** result, since both does not employ SSE optimization. In other words, **poptrie** may

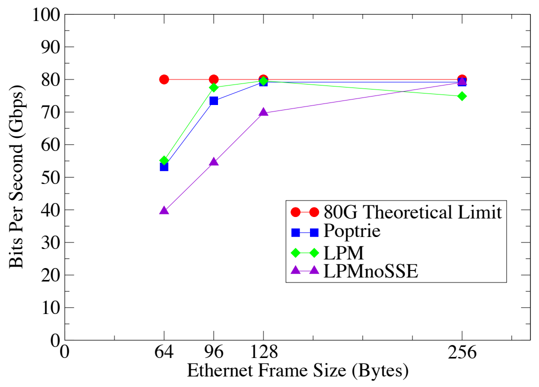

Figure 4: Random traffic on ntt (bps).

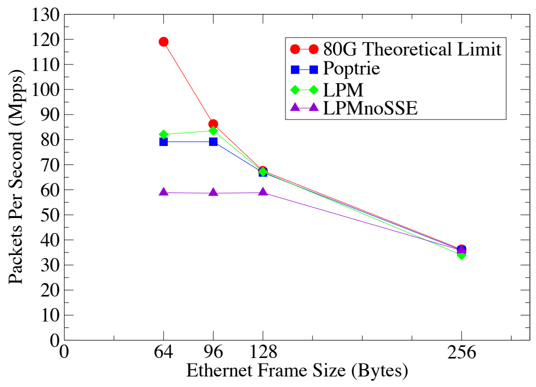

Figure 5: Random traffic on ntt (pps).

benefit from SSE optimization to obtain a better performance in the future.

Aside from the fair evaluation, LPM with SSE optimization also succeeded in performing the wire-rate. However, some drawbacks seem to exist in the LPM; 1) the memory used by LPM in the case of larger routes are prohibitive, and 2) the loading of the large routes in the routing table in LPM is slow. The latter is because LPM sequencially searches the prefix range in the array. Poptrie did not seem to inherit such issues.

6. CONCLUSION AND FUTURE WORK

This paper describes the implementation of a PC based high-speed software router which can achieve 80 Gbps. With standard linux with small modification using a new routing lookup algorithm called Poptrie, PC hardware for high performance server can act as a high speed router for quite reasonable costs. The paper also evaluates the performance and practical aspects of the implementations.

The highest packet-transmitting speeds of optical fiber circuits are reaching 400Gbps per laser beam, today. In such an environment with speed exceeding 100Gbps, ASIC based forwarding technology is still quite essential and hardware forwarding system is still critical to minimize RTT in the network which is quite important to carry especially TCP based protocol such as HTTP in an efficient way. So we do not think that we can replace all the routers and/or switches by software based counterparts entirely. However, software based high-speed router can change the landscape of the network edge such as datacenters which collocate the servers, and the access network with high speed CPE (Customer Premises Equipment).

7. REFERENCES

[1] H. Asai and Y. Ohara. Poptrie: A compressed trie with population count for fast and scalable software ip routing table lookup. SIGCOMM '15, pages 57–70. ACM, 2015.

[2] M. Chiosi et al. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Introductory white paper, ETSI, 2012.

[3] David Meyer. University of Oregon Route Views Archive Project. `http://routeviews.org/`.

[4] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. SOSP '09, pages 15–28. ACM, 2009.

[5] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM '98. Proceedings. IEEE*, volume 3, pages 1240–1247 vol.3, Mar 1998.

[6] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: A gpu-accelerated software router. SIGCOMM '10, pages 195–206. ACM, 2010.

[7] Intel. DPDK: Data Plane Development Kit. `http://dpdk.org/`.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[9] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. Clickos and the art of network function virtualization. NSDI'14, pages 459–473, 2014.

[10] PCI SIG. PCI Express base specification 3.0.

[11] Radisys. Benchmark Study — October 2014. `http://go.radisys.com/rs/radisys/images/2014-10-Paper-DPDK-Benchmark.pdf`.

[12] L. Rizzo, M. Carbone, and G. Catalli. Transparent acceleration of software packet forwarding using netmap. In *INFOCOM, 2012 Proceedings IEEE*, pages 2471–2479, March 2012.