Kamuee: An IP Packet Forwarding Engine for Multi-Hundred-Gigabit Software-based Networks

Yasuhiro Ohara[a], Hiroki Shirokura[a], Abhik Datta Banik[a], Yudai Yamagishi[a], Kim Kyunghwan[b]

[a]*NTT Communications Corporation*
[b]*Independent Engineer*

Abstract

In Software Defined Network (SDN) and Network Function Virtualization (NFV) era, extensible, flexible, and yet high-performance software packet forwarding capability is desirable, as the core functionality of the future Internet. In this paper we present Kamuee: a software IP packet forwarding engine, and its core router version that builds only on commodity hardware. By scaling the number of forwarding CPU cores, Kamuee supports multiple 100GbE interfaces, with near wire-rate traffic. In our benchmarks that use two Intel Xeon Platinum 8180 processors, Kamuee could forward 349.60 Gbps 512B-sized random destination traffic in a BGP full route environment. Further intralaboratory evaluation resulted in 292.17 Mpps 64B forwarding capability, showing the potential of very high-speed software router.

Keywords: NFV, SDN, Software Router, DPDK

1. Introduction

The rapid emergence of services based on technologies such as cloud computing and AI-driven robotics, underlines the desperate requirement for next generation networks, which are more flexible and able to keep pace with changes in usage and capacity. Software-defined networking (SDN), network functions virtualization (NFV), and network virtualization (NV) are the solutions which address this requirement and provide novel methodologies to design, build and operate next generation networks. Thanks to the recent breakthroughs like off-the-shelf hardware or whitebox networking, a massive paradigm shift in networking technology finally took place by decoupling software from the hardware, so that it is no longer constrained by the box that delivers it. SDN and NFV have become indispensable for all telecommunication service providers to (1) ***Drive Innovation*** by creating new types of applications, on-demand services and business models (2) ***Deliver Agility and Flexibility*** by enabling rapid deployment of new applications, services and infrastructure to quickly fulfill their changing requirements (3) ***Reduce OpEX*** by enabling automation and algorithm control through increased programmability of network elements to make it simple to design, deploy, manage and scale networks and (4) ***Reduce CapEx*** by enabling network functions to run on commodity off-the-shelf (COTS) hardware.

Therefore, software-defined networking is not only good for the network, but for the business as well. With SDN, the network can be made programmable. A deeper look into network programmability reveals that it involves both the control plane and the data plane and that both are valuable in containing costs and enabling business growth. Data path programmability offers a platform for rapidly deploying a new service or modifying an existing one. This is particularly important to service providers in fields like security, for example, to mitigate a massive DDoS attack affecting hundreds of vulnerable servers across the globe.

Today, both control and data plane programmability provided by software-based solutions are the desirable characteristics of network services and devices. An integral part of this is software based processing of data packets. But without high performance, software packet processing cannot leverage the potential benefits of SDN and NFV. Recent unprecedented leaps in NFV with performance reaching up to 100 Gbps [1, 2], make the need for software routers supporting speeds above 100 Gbps even more imperative. Keeping this in mind, we endeavored to develop a high performance software IP packet forwarding engine called ***Kamuee*** with comprehensive Layer 3 (routing) functionalities. Kamuee

successfully integrates the capabilities of DPDK (Data Plane Development Kit) [3] with Poptrie [4] to achieve stable and reliable high speed software IP packet routing. Kamuee is a fully fledged software router that can run on COTS PC Server and is capable of forwarding 344.28 Gbps random destination IP traffic with Ethernet frame size greater than 512B and approx. 600,000 full routes.

In this paper we present the design, implementation and evaluation of Kamuee. Section 2 describes the related research. Section 3 highlights the design and implementation methodology of Kamuee. Section 4 elaborates the performance evaluation of Kamuee. Kamuee was used as one of the core routers in Interop Tokyo 2018 and the relevant experience has been shared in section 5. Section 6 concludes the paper.

2. Related work

Achieving high performance in software routing implementations has been a major challenge over the last two decades and has lead to some cutting-edge advances in this field. Slowness of Linux network stack performance has become increasingly relevant issue over the years because of the exponential increase in the amount of data that is being transferred over networks and the corresponding workloads. Even the widespread use of 10 GbE network cards could not resolve this issue because of some bottlenecks in Linux kernel itself that prevent packets from being quickly processed. There have been many attempts to circumvent these bottlenecks with kernel bypass techniques that enable packet processing without involving the Linux network stack such that the application running in the user space communicates directly with networking device. Intel's DPDK [3] is one such solution which takes care of the packet forwarding performance bottleneck. DPDK leverages existing Intel Processor technologies like SIMD instructions (Singles Instruction Multiple Data), Huge-pages memory, multiple memory channels and caching to provide packet processing acceleration with its own libraries. Recent innovation like Poptrie addresses another major bottleneck of IP routing lookup. Poptrie [4] leverages the population count instruction to give the indirect indices to the descendant nodes in order to keep the small memory footprint within the CPU cache and enables extremely high speed IP lookup. Kamuee harnesses (1) packet processing acceleration of DPDK (2) high-speed packet lookup provided by Poptrie and (3) parallel processing for its superior performance. Read-Copy Update (RCU) [5] has been used to achieve the latter, for concurrency control owing to its lock/synchronization specialization.

Kamuee-Zero [6] presents the routing table mechanisms of the previous version of this implementation. The paper shows that 1) in order to achieve near wire-rate performance in 40GbE, more than three queues per port are necessary, 2) the difference in throughput performance between NUMA-aware and NUMA-nonaware is not large (if not negligible), and 3) performance exceeding one hundred Gbps can be achieved using software based router, with four 40GbE interfaces and 128B short packet random traffic. Kamuee-Zero did not have routing capability as it did not support any routing protocol. In contrast, Kamuee (the version presented in this paper) supports all major routing protocols such as BGP, OSPF, and RIP, and other required functions that are necessary to function as a basic router, such as ARP, VLAN, and statistics counter. Also, while Kamuee-Zero supports only 40GbE network hardware, Kamuee can additionally support 100GbE hardware as well.

While NFV tackles the problems posed by legacy proprietary middleboxes [7] by leveraging virtualization technologies to implement network functions (NFs) on commodity hardware, the advantages of NFV come with some downsides [8] as software-based NFs can potentially introduce significant performance overheads. Several research works have been done to address the performance drawback of software based NFV. ClickNP [9] offloads software logic onto programmable hardware like FPGA to accelerate individual NFs. NetBricks [10] runs NFs on a single CPU core instead of virtual machines and containers to improve NF performance. ClickOS [11], DPDK [3] and NetVM [12] optimize and accelerate packet processing from the network hardware to and between virtual machines. Recently, there have been some astounding progress in NFV performance in the past couple of years. Metron [1] realizes high performance NFV service chains at the emerging and extremely challenging link speeds at 100 Gbps using commodity hardware, while significantly reducing latency. Andromeda [2], Google Cloud Platform's network virtualization stack demonstrates that an OS bypass software data-path provides performance competitive with hardware, achieving 32.8Gb/s using a single core. Some other advancements in this area include SafeBricks [13] protecting NFs in cloud environments, a novel programming interface forNon-Volatile Main Memory called PASTE [14] , NFV resource manager ResQ [15] supporting high performance in multi-tenant NFV clusters, and, highly scalable and resilient general purpose L2 switching software FBOSS [16] capable of

running on commodity hardware.

3. Design and implementation

The main motivation of Kamuee design is the following: if the software is good and simple, we should be able to get a good performance out of a good hardware. Furthermore, we should be able to increase the overall performance by adding more hardware resources (i.e., CPU cores). To achieve this, Kamuee was designed and implemented as simple as possible, in the belief that any complexity may lead to performance degradation (i.e., KISS principle).

Kamuee employs the run-to-completion model [17] rather than pipe-line model: we adopted this model because the run-to-completion model is suggested as a better-performance model in a past work called RouteBricks [18]. By employing the run-to-completion model, we can scale-out the packet forwarding process over the multiple CPU cores, enabling the design goal of increased performance by increased hardware.

Some design policies are inspired by others' work. RCU is utilized in Linux kernel and also in some DPDK applications [19]. Use of Tap devices [20, 21] or KNI interface [22] to map the physical NICs in Linux to connect to the open-source software, is a well-known approach and is also supported by VPP [23].

CPU core (equivalently, CPU time) is a very precious resource for our purpose: if we have spare CPU core, we could gain more performance. Thus, we assemble the slow tasks (i.e., the tasks that do not need high performance), such as netlink, RIB, acl, tap, arp, and vty (i.e., virtual terminal), on one CPU core, to avoid wasting CPU core. To achieve this, we used Lthread library [24] included in the DPDK source (located under `examples/performance-thread`). Furthermore, since the function call interface are the same, we can change native DPDK thread to lthread thread and vice versa, enabling balance in performance. For example, because our experience suggested to speed them, in our current recommended setting, rib-manager, tap-manager, and snmp-manager run as the native DPDK threads, consuming one CPU core for each.

Figure 1 illustrates the internal structure of the Kamuee software router. In the fundamental DPDK concepts, Network Interface Cards (NICs) belongs to either Linux space or DPDK space. The figure happens to show the case that two NICs belong to Linux, and four NICs belong to DPDK, but it is configurable. The packets received in DPDK NICs are distributed to a specific CPU core by the RSS/multiqueue technology of the NIC. The assigned CPU core is running a "forwarder"

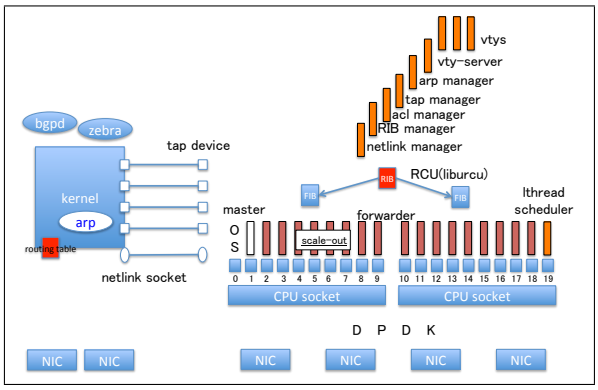


Figure 1: Kamuee internal structure: the ordinary linux space with two NICs and the DPDK space with four attached NICs are illustrated on left and right, respectively. Two CPU sockets are shown on the right, with most of the CPU cores running "forwarder" DPDK thread, enabling *scale-out* of packet forwarding tasks on the multiple cores. The routing table (RIB) is compiled as FIB using Poptrie, and distributed to each CPU socket's L3 cache using RCU. The manager threads such as ones dealing with netlink, tap, and vty, are launched on the right-most core using the "lthread" library. DPDK-attached physical Kamuee NICs are 1-to-1 mapped on the tap devices in the Linux space so that the routing protocol open-source software (e.g., Quagga) can run on the Kamuee NICs. The routes calculated by the routing daemons are first installed in the Linux kernel, and then propagated to Kamuee RIB using netlink socket/messages.

thread of Kamuee, that is built around the DPDK thread and occupies the CPU core. The packet, if it is to be forwarded, is solely handled by that CPU core, without the need of using the other CPU cores; hence the run-to-completion model. The forwarder thread processes the protocol headers such as Ethernet and IP, looks up the routing table (labeled "FIB" in the figure), and then directly forward the packet to the other NIC to emit the packet. Thanks to the Poptrie [4] that can compress some hundreds of thousands of routes into a few megabytes memory footprint, the routing table lookup can be completed in some CPU L3 cache accesses, avoiding a lot of main memory accesses. This simple run-to-completion forwarding process without main memory access, together with the parallelization (in other words, scale-out) on the number of CPU cores, is the main reason of high-performance of Kamuee. This is the key design of Kamuee, and is the main contribution of this paper.

The control protocol packets and the self-destined packets (i.e., the packets that are to be received by the Kamuee host itself) are handled as follows: by routing table lookup in the forwarder, the packet is indicated to be passed to the Linux part of the system. Then, the packet is passed to the "tap manager" (shown top-right in Figure 1). The tap manager delivers the packet to the

```
0.0.0.0/2 nexthop: 192.85.1.3 port: 2 flags:
64.0.0.0/2 nexthop: 193.85.1.3 port: 0 flags:
128.0.0.0/2 nexthop: 194.85.1.3 port: 8 flags:
192.0.0.0/2 nexthop: 195.85.1.3 port: 6 flags:
```

Figure 2: Four default routes or "`default4`"

Linux kernel via the correspondent tap device that is 1-to-1 mapped to the receiving NIC. In this way, the Linux kernel, and thus the Linux user processes, can receive the packet from DPDK attached NICs without problem. The routing protocol daemons use this mechanism: for example, Quagga bgpd (shown top-left in the figure) receives the BGP packets, processes them, and installs the calculated BGP routes in the Linux kernel. The newly installed routes are notified through the Netlink mechanism to the Kamuee's "netlink manager" (shown top-right in the figure). The "rib manager" is informed of the new routes by the netlink manager, and the rib manager installs it in the main routing table (labeled "RIB" in the figure), and produces the FIB using the Poptrie algorithm. One FIB for each CPU socket is prepared to properly support the CPU cache mechanism.

4. Evaluation

4.1. Benchmark setup

In our benchmark method, we measure the performance of the Device-Under-Test by sending the random-destination IP traffic from the network test: Spirent, by having the DUT forward back the traffic, and then by counting the packets returned to the network tester. To achieve this, we install in the DUT four prefixes that cover all IPv4 address space, so that the DUT can return all the packets that it could forward, back to the Spirent tester machine, without causing "route not found" error. We refer to the four prefixes that are shown in Figure 2 as "Four default routes" or "default4".

Wherever BGP full routes are used in the benchmark, the snapshot taken in NTT Communications' TestBed on 2016/12/12 has been used, in addition to Four default routes. The snapshot includes 612,916 prefixes.

If not specified explicitly, the compiler optimization option defaults to "-O3".

We generally focus on the achieved bandwidth rather than transaction performance (i.e., packet per second or pps). The factor limiting the performance is generally either transaction performance or the bandwidth of subsystems, such as CPU core, QPI, and PCIe. Since we want to understand the overall performance of the system as a whole, and since the transaction performance can be calculated from the bandwidth performance, we generally focus on bandwidth performance, unless we have a specific interest.

Table 1: Hardware specification of Kamuee

Hardware Type	Product Name
Chassis	SYS-7049GP-TRT
Motherboard	Supermicro X11DPG-QT
CPU	Intel Platinum 8180 ×2
Memory	DDR4-2133 16GB ×12 = 192GB
NIC	Mellanox Connect-X5 100GbE Dual-Port ×5
	Intel X710 10GbE Quad-Port ×1

Table 2: Softwares used in Kamuee

Software Package	Version
OS	Ubuntu 16.04.5 amd64
DPDK	17.11
userspace-rcu	0.9.3
Quagga	0.99.24
net-snmp	6.0.1-2

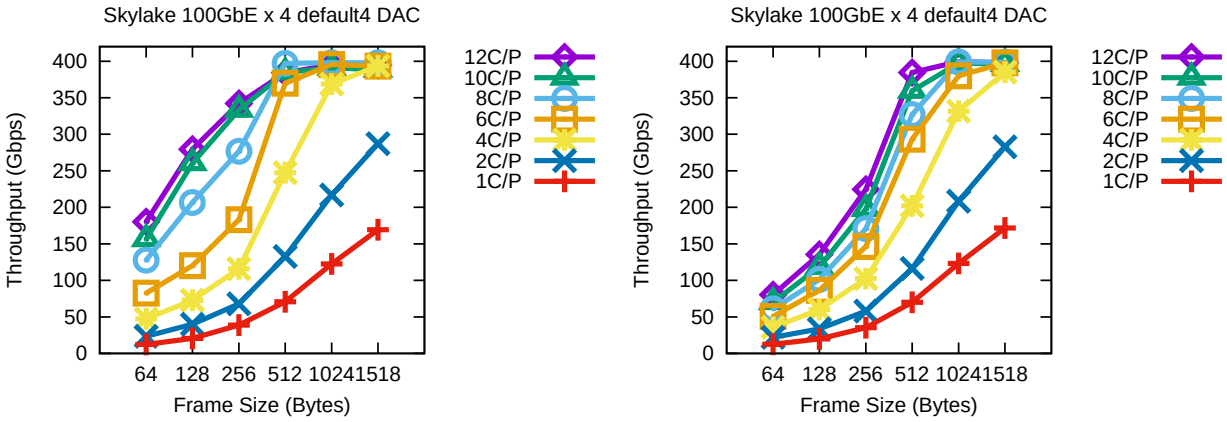
Our version of the software demonstrated a significant fluctuation of the performance over time. We have measured the performance as the average of five samples. It should be noted that our measurement may not cover the significant period of the fluctuation. Sometimes, after 30-40 seconds, Kamuee exhibits performance degradation from 394 Gbps to 383 Gbps in the same setting (this is shown later in Figure 4).

Table 1, 2 lists the hardware and software specification, respectively. Quagga was used to provide BGP4/4+, OSPF and OSPFv2 functionalities on Kamuee.

4.2. Overall Throughput in Bandwidth (BPS)

Figure 3a shows the bandwidth throughput performance of Kamuee when given wire-rate random traffic from all four 100GbE interfaces. The figure compares the performance on one core per port (1C/P) through twelve cores per port (12C/P). It shows that Kamuee successfully increases its performance as the number of CPU cores increase. With 64B shortest Ethernet frame, one core/port setting and twelve cores/port setting exhibit 12.20 Gbps and 180.18 Gbps, respectively. With 1518B longest frame, 12C/P demonstrate almost wire-rate of 394.36 Gbps.

The setting of Figure 3a is normal Linux connected routes plus "`default4`", leaving the Local Loopback Address 127/8 destined to the Linux host's upper layer.



(a) LoopbackTap (b) LoopbackBlackhole

Figure 3: Throughput with/without Tap Route

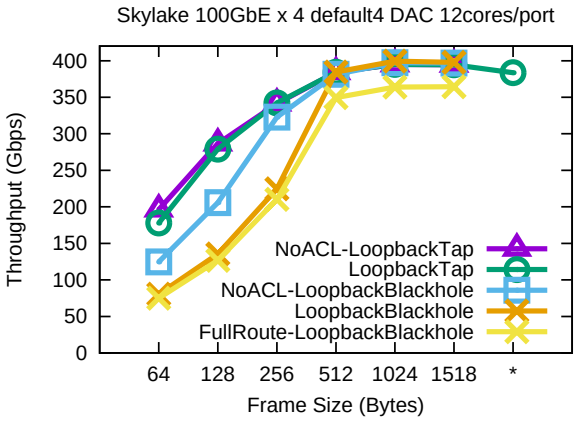


Figure 4: With/Without Tap Route. After 30 or 40 seconds with 1518B 400 Gbps traffic, With-Tap-Route degrades its performance slightly (shown in * mark in the x-axis). Except the one labeled with `FullRoute`, the route-setting is `default4`.

Since we launch fully-random-destination traffic from the tester to the Kamuee, the traffic destined to 127/8 fills and overloads the TAP socket, and sometimes cuts the BGP session through it. To avoid such problem, we installed 127/8 as blackhole route, and unexpectedly it lowered the throughput performance for some unknown reason. The performance is shown in Figure 3b. Since traffic without the TAP destined ones are more realistic, this can be deemed as the real performance value. We can infer the root cause of the performance degradation such that some bug or problem lies in Kamuee or DPDK library. In either way, Figure 3a can be recognized as the potential value of the software router: no matter what bug the root cause is, the performance value of Figure 3a shows the potential capability of the software router, if we trust the reliable Spirent tester.

Figure 4 compares the two settings of Local Loopback Address that are either directed to TAP or Blackhole. With four 100GbE I/Fs, twelve cores per port is the best setting we can get, and the figure only shows the case. 64B case degrades from 180.18 Gbps (at LoopbackTap) to 80.52 Gbps (at LoopbackBlackhole).

We integrated ACL functionality into Kamuee, by incorporating the DPDK ACL library. Our ACL implementation in the benchmark test did not include *ANY* entry, and just searched in the ACL list only to find the default ACL entry. Even in that setting, the ACL function exhibits a significant performance degradation. We show the overhead of the ACL functionality: NoACL-LoopbackBlackhole outperforms LoopbackBlackhole significantly. This suggest that by improving the ACL functionality, we may be able to reduce the performance degradation in the future.

Also we show the impact of FullRoute: comparing LoopbackBlackhole (equipped with `default4`) and FullRoute-LoopbackBlackhole, we can see the impact of holding and looking up the BGP full routes is not large, thanks to the Poptrie technology.

4.3. Overall Throughput in Transaction (PPS)

Figure 5a shows the transaction performance values in Packet Per Second (PPS), in contrast to the theoretical limit labeled as “Wire-rate”. Figure 5b illustrates the achievement rate against the ideal wire-rate. In 64B case, the most realistic setting, i.e., FullRoute-LoopbackBlackhole, exhibits 111.42 Mpps (achievement rate: 0.19), while NoACL-LoopbackTap for refer-

ence shows 292.17 Mpps (achievement rate: 0.49). The NoACL-LoopbackTap's reference value is not a bad value, but the realistic FullRoute-LoopbackBlackhole is not surprisingly fast, and gives a moderate speed performance value.

4.4. Effect of compiler optimization

Figure 6 shows the effect of compiler optimization on the performance of LoopbackTap setting. It demonstrates that gcc optimization option -O1 and above are roughly the same performance when the gcc version is (Ubuntu 5.4.0-6ubuntu1 16.04.9) 5.4.0 20160609. Further discussion such as which optimization option impacts the most is future work.

4.5. Microflow: the Benchmark of a Single IP Flow

Table 3 gives a list of latency measurements that is conducted for five minutes or more. Microflow means the single IP session flow, so the RSS (Receive-Side Scaling) of NIC cannot split the traffic onto multiple cores. We have two NUMA types (Same or Cross, meaning whether the test traffic needs to come accross the different NUMA nodes), and six Ethernet frame sizes.

Overall, around 5 Mpps is performed for the single core forwarding performance. No visible difference in latency was observed between NUMA types of the traffic. The latency is in average 20-30 microseconds for the Ethernet frame longer than 512B, but it was larger (such as 254 and 333 microseconds) in the Ethernet frame shorter than 256B. We suspect that the effect of PCI's Max Read Request Size is involved [25].

4.6. Packet loss

10 Gbps traffic was measured to test the packet loss rate in the not-so-heavy traffic load. This time the IP destination address field is randomized so that the RSS of NIC can split traffic to multiple cores. The 10 Gbps traffic was forwarded without major problem regardless of NUMA type. For the duration of five minutes, a rather small number of frames are dropped such as less than 10,000 frames (Table 4). It suggests the packet loss rate is significantly low to support the real traffic.

4.7. Benchmark for Virtualized Function

In order to investigate the bandwidth performance of Kamuee in virtual environment, VMs have been created using KVM with the same configuration as physical environment. Benchmarking has been done using similar test environment and default4 routes as described earlier. The VMs set up virtual CPUs with similar NUMA configuration and number of cores as the physical environment, with each virtual core using "vcpupin" such that the virtual cores do not operate on the same physical core. NICs are directly connected with SR-IOV using PCI-passthrough.

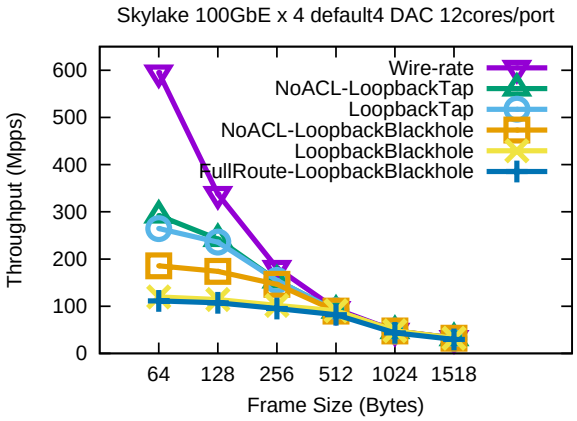
Figure 7 compares the performance of two cores per port (2C/P) through twelve cores per port (12C/P). Here, for packets longer than 1024B, a strange phenomenon is observed: the performance improves on reducing number of cores. The cause of this phenomenon can be IOMMU. The intel_iommu is an option related to DMA for enabling PCI-passthrough in Intel CPUs. Further investigation of the difference in performance due to the presence or absence of intel_iommu=on revealed that performance always degrades when IOMMU is effective regardless of bare-metal or virtual environment. This is illustrated in Figure 8.

5. Experience in Interop Tokyo 2018

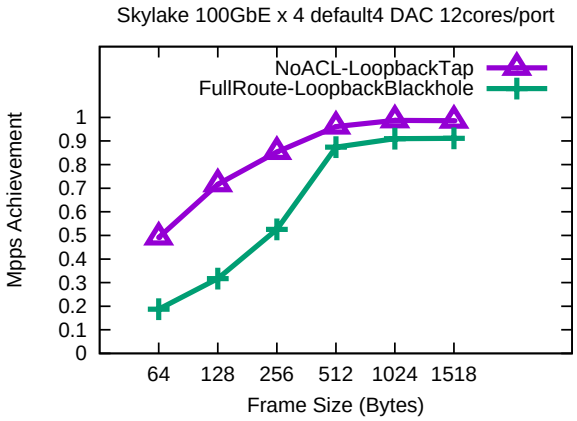
We deployed Kamuee as a backup core router in the backbone network of Interop ShowNet 2018. Interop ShowNet has an experimental Internet backbone network deployed during the three-day Interop event. This is one of the largest experimental networks and every year, many product vendors bring in their new products to test their performance and interoperability. This year, over 2600 devices and services were connected to the network and over 450 engineers participated to build this network. As a backup core router of the network, Kamuee was responsible for forwarding all the traffic of this large scale network in case the primary core router fails.

The hardware specification used for Kamuee in Interop ShowNet 2018 is the same as the one in the previous benchmark, and shown in Table 1. Total of six routers were connected directly to Kamuee; four connected using 100GbE-SR4, one connected using 100GbE-LR4, and one connected using 10GbE-LR. The one using 10GbE-LR was the route reflector. Additionally, two network traffic generators were connected, one connected using 100GbE-LR4, and another connected using 10GbE-LR.

Kamuee was configured to provide the best performance for user traffic within the hardware constraints, such as limited number of CPU cores and NICs. We allocated eight cores per port to forwarder threads for 100GbE ports connected to the primary core router and the backup aggregation router. As we had limited number of CPU cores, we chose to only allocate six cores per port to forwarders for 100GbE ports connected to the AS border routers and the network traffic generators



(a) PPS throughput



(b) PPS achievement rate

Figure 5: Throughput in PPS

Table 3: Microflow Latency

			Avg. over 5 samples		mm:ss		Latency (us)		
NUMA	Cable	fsize	Tx load	Rx throughput	Time	#rx-frames	Min.	Avg.	Max.
Same	DAC	64	100Gbps/148.8Mpps	3.53Gbps/5.26Mpps	5:18	1,587,401,784	16.66	69.4	692.25
Same	DAC	128	100Gbps/148.8Mpps	5.63Gbps/4.76Mpps	6:20	1,823,246,252	10.43	71.95	120.82
Same	DAC	256	100Gbps/148.8Mpps	8.71Gbps/3.94Mpps	6:31	1,542,903,228	12.49	253.73	884.24
Same	DAC	512	100Gbps/148.8Mpps	21.49Gbps/5.05Mpps	5:34	1,670,079,851	4.86	22.01	112.36
Same	DAC	1024	100Gbps/148.8Mpps	38.02Gbps/4.55Mpps	8:21	2,378,373,317	5.86	22.18	892.35
Same	DAC	1518	100Gbps/148.8Mpps	61.19Gbps/4.97Mpps	5:54	1,696,514,167	10.87	34.23	184.03
Cross	DAC	64	100Gbps/148.8Mpps	3.32Gbps/4.94Mpps	7:16	2,220,846,968	16.35	69.46	859.5
Cross	DAC	128	100Gbps/148.8Mpps	5.88Gbps/4.97Mpps	7:01	2,089,926,340	12.18	75.63	2,183.4
Cross	DAC	256	100Gbps/148.8Mpps	6.40Gbps/2.90Mpps	5:12	903,051,580	12.02	333.07	369.8
Cross	DAC	512	100Gbps/148.8Mpps	22.07Gbps/5.19Mpps	11:55	3,661,038,680	4.85	22.06	64.8
Cross	DAC	1024	100Gbps/148.8Mpps	39.93Gbps/4.78Mpps	8:05	2,309,901,569	6.57	22.63	950.93
Cross	DAC	1518	100Gbps/148.8Mpps	51.56Gbps/4.19Mpps	12:15	3,194,555,373	9.79	25.37	812.38

as these ports only forwarded limited number of traffic. As 10GbE ports do not require many cores to provide performance, we only allocated two cores per port to forwarders responsible for the 10GbE ports. In total, fourty-six cores were used as forwarders to forward the traffic.

As a core AS router, the routing table of Kamuee consisted of the Internet full routing table and AS internal routes. Over 700K routes were registered on the IPv4 routing table while IPv6 routing table and had only about 59K routes.

From our experience of operating Kamuee in Interop ShowNet 2018, we discovered the following key problems that need to be addressed when running software routers as core routers using COTS devices:

- Nonoptimal cooling inside the chassis
- Differences in NIC implementation per vendor

First problem we encountered was excessive heating of the NICs. Unlike specialized networking chassis which have optimized cooling for NICs and network processors, the COTS server chassis used for Kamuee was not equipped to do so. The temperature sensor readings showed temperature of up to 67 degrees centigrade while the maximum operating temperature for100GbE-LR4 QSFP28 module is 70 degrees centigrade [26]. This calls for some additional cooling design while using COTS hardware as high speed network device to avoid errors due to overheating.

Second problem we faced is the difference in implementations of NIC functionalities and their corresponding DPDK drivers per vendor. Kamuee's initial design used KNI as the packet interface between the dataplane and kernel. Though it worked fine for NICs from a single vendor, it malfunctioned when we started to use NICs from multiple vendors simultaneously. We needed to switch back to the slower Tun/Tap kernel interface to address the malfunctioning problem.

Table 4: 10Gbps traffic packet loss

Avg. over 5 samples mm:ss

NUMA	Cable	fsize	Tx load	Rx throughput	Time	#tx-frames	#rx-frames	#loss*	loss-rate+
Same	DAC	64	10Gbps/14.88Mpps	10.00Gbps/14.88Mpps	5:22	4,793,907,953	4,793,899,029	8,912	1.86e-06
Cross	DAC	64	10Gbps/14.88Mpps	10.00Gbps/14.88Mpps	5:13	4,666,135,849	4,666,135,251	592	1.27e-07

* The tester's tx, rx, and loss counts didn't seem to be consistent for unknown reason.
+ Loss-rate is calculated by dividing #loss by #tx-frames.

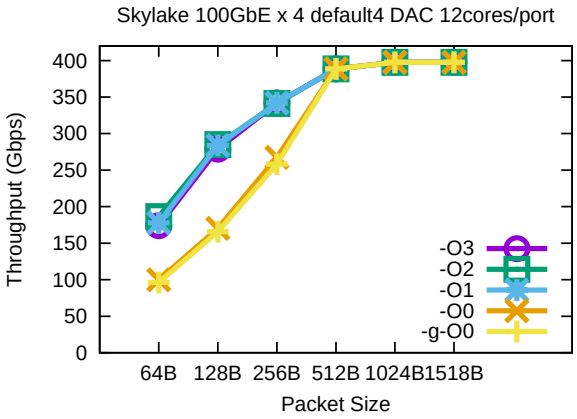


Figure 6: Compiler Effect

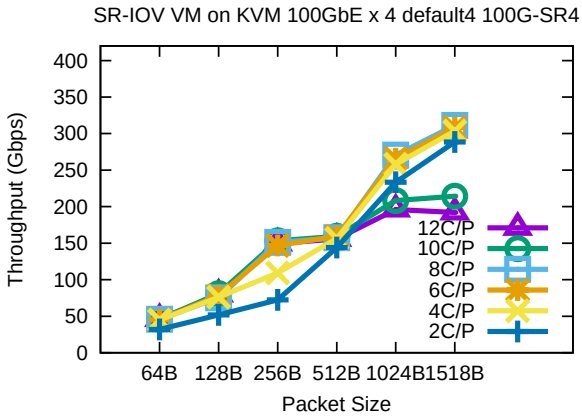


Figure 7: KVM cores per port effect

Kamuee is the first ever software router to be used in Interop Shownet backbone as one of the core routers, and throughout the event Kamuee ran properly without any glitches or errors. Kamuee lived up to the challenge of a fully functional software router interoperable with most of the world's leading router vendors, namely, Cisco, Juniper, and Huawei, and this is a promising stepping stone for its future commercial success as a mainstream software router.

6. Conclusion and future work

We revealed the tricks for high-performance software router design: we think the run-to-completion model in DPDK, the Poptrie algorithm, and the keep it as simple as possible principle, are the main grounds for our superior performance.

We show a good performance benchmark value of Kamuee: for 512B-sized frames or longer, Kamuee can forward around 349.60 Gbps random destination traffic with BGP full routes (at FullRoute-LoopbackBlackhole in Figure 4). Further intralaboratory evaluation of 12C/P at 64B frame in LoopbackTap (Figure 3a), the Kamuee showed the potential of forwarding 292.17 Mpps.

We have seen many unstable characteristics of Kamuee, that might be common to the general software router. The throughput performance fluctuated over time, in a significantly large range of a few tens of Gbps. Further, we sometimes saw performance degradation that we cannot explain (yet). Since the performance of the software router is very high and promising, the need to address the aforementioned drawbacks becomes even more imperative.

Even with some drawbacks, our Interop experience proved that Kamuee satisfies the necessary functions and quality needed to sustain the large scale IP infrastructure. Overall, Kamuee demonstrated a promising performance for the use of future virtual network functions in the NFV environment.

As an open problem for the future network, the algorithms for the Access Control List (ACL) and firewall applications that maintain high performance even with some hundreds of thousands to some millions of ACL entries are the next challenge in this field.

References

[1] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, G. Q. M. Jr., Metron: NFV service chains at the true speed of the underlying hardware, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX

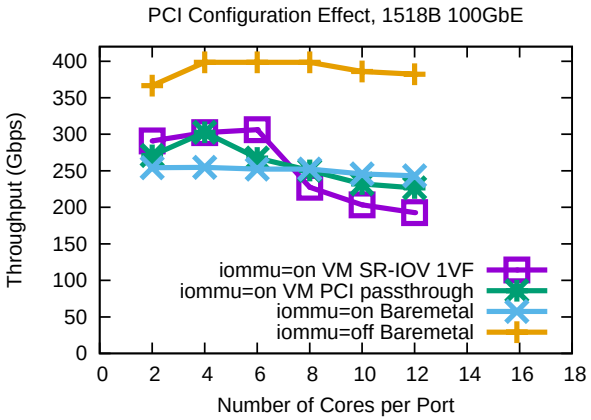


Figure 8: PCI configuration effect

Association, Renton, WA, 2018, pp. 171–186.
URL https://www.usenix.org/conference/nsdi18/presentation/katsikas

[2] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer, J. Alpert, J. Ai, J. Olson, K. DeCabooter, M. de Kruijf, N. Hua, N. Lewis, N. Kasinadhuni, R. Crepaldi, S. Krishnan, S. Venkata, Y. Richter, U. Naik, A. Vahdat, Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 373–387.
URL https://www.usenix.org/conference/nsdi18/presentation/dalton

[3] Intel, DPDK – Data Plane Development Kit, http://dpdk.org/.

[4] H. Asai, Y. Ohara, Poptrie: A compressed trie with population count for fast and scalable software ip routing table lookup, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, 2015.

[5] P. E. McKenney, J. D. Slingwine, Read-copy update: Using execution history to solve concurrency problems, in: Parallel and Distributed Computing and Systems, 1998, pp. 509–518.

[6] Y. Ohara, Y. Yamagishi, Kamuee zero: the design and implementation of route table for high-performance software router, in: Proceedings of Internet Conference 2016, IC 2016, 2016.

[7] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else's problem: Network processing as a cloud service, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, ACM, New York, NY, USA, 2012, pp. 13–24. doi:10.1145/2342356.2342359.
URL http://doi.acm.org/10.1145/2342356.2342359

[8] N. W. Paper, Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action. issue 1 (Oct 2012).

[9] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, E. Chen, Clicknp: Highly flexible and high performance network processing with reconfigurable hardware, in: Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 1–14. doi:10.1145/2934872.2934897.
URL http://doi.acm.org/10.1145/2934872.2934897

[10] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, S. Shenker, Netbricks: Taking the v out of NFV, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, Savannah, GA, 2016, pp. 203–216.
URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/panda

[11] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, Clickos and the art of network function virtualization, in: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 2014.

[12] J. Hwang, K. K. Ramakrishnan, T. Wood, Netvm: High performance and flexible networking using virtualization on commodity platforms, IEEE Transactions on Network and Service Management 12 (1) (2015) 34–47.

[13] R. Poddar, C. Lan, R. A. Popa, S. Ratnasamy, Safebricks: Shielding network functions in the cloud, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 201–216.
URL https://www.usenix.org/conference/nsdi18/presentation/poddar

[14] M. Honda, G. Lettieri, L. Eggert, D. Santry, PASTE: A network programming interface for non-volatile main memory, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 17–33.
URL https://www.usenix.org/conference/nsdi18/presentation/honda

[15] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, S. Shenker, Resq: Enabling slos in network function virtualization, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 283–297.
URL https://www.usenix.org/conference/nsdi18/presentation/tootoonchian

[16] S. Choi, B. Burkov, A. Eckert, T. Fang, S. Kazemkhani, R. Sherwood, Y. Zhang, H. Zeng, Fboss: Building switch software at scale, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18, ACM, New York, NY, USA, 2018, pp. 342–356. doi:10.1145/3230543.3230546.
URL http://doi.acm.org/10.1145/3230543.3230546

[17] 8. Poll Mode Driver – Data Plane Development Kit 18.08.0 documentation, http://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html.

[18] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, S. Ratnasamy, Routebricks: Exploiting parallelism to scale software routers, in: Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSP '09, 2009.

[19] S. Hemminger, Making a virtual router a reality with dpdk, rcu and 0mq, https://events.static.linuxfound.org/sites/events/files/slides/DPDK_RCU_0MQ.pdf.

[20] Universal TUN/TAP device driver., https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

[21] 33. Tun—Tap Poll Mode Driver – Data Plane Development Kit 18.08.0 documentation, https://doc.dpdk.org/guides/nics/tap.html.

[22] Ferruh Yigit, Interworking with the Linux Kernel, https://dpdksummit.com/Archive/pdf/2016Userspace/Day02-Session06-FerruhYigit-Userspace2016.pdf.

[23] VPP Sandbox/router - fd.io, https://wiki.fd.io/view/VPP_Sandbox/router.

[24] Hasan Alayli, lthread, a multicore enabled coroutine library written in C, https://github.com/halayli/lthread.

[25] Y. Ohara, Y. Yamagishi, S. Sakai, A. D. Banik, S. Miyakawa, Revealing the necessary conditions to achieve 80gbps high-speed pc router, in: Proceedings of the Asian Internet Engineering Conference, AINTEC '15, 2015.

[26] Mellanox, Mellanox 100GbE QSFP28 LR4 Optical Transceiver, https://www.mellanox.com/related-docs/prod_cables/PB_MMA1L10-CR_100GbE_QSFP28_LR4_Transceiver.pdf.