Kamuee Zero: the Design and Implementation of Route Table for High-Performance Software Router

Yasuhiro Ohara[a], Yudai Yamagishi[b]

NTT Communications Corporation

[a] *yasuhiro.ohara@ntt.com*
[b] *y.yamagishi@ntt.com*

Abstract

We desire, in general, a flexible, controllable intelligent communication network infrastructure, that can robustly support human activities in a large scale. Toward such SDN/NFV oriented network design, we start to seek for a good software router design in this paper.

We introduce a software router implementation called "Kamuee", which integrates DPDK, RCU, and Poptrie. In this paper the design principle, the architecture, and the software implementation status are explained. We show through the performance evaluation that 1) the performance figure of some hundreds giga bps is achievable by the use of commodity hardware, 2) the packet loss problem still remains as the severe problem to overcome, 3) Poptrie can handle very high-speed routing decisions for 40GbE wire-rate traffic, actually in a real IP router system.

Kamuee implementation and its simple design principle are expected for various further applications. Examples include the development of high-speed middleboxes such as firewalls, CGNs, load-balancers, or massive scale routers handling quite a number of BGP and/or VPN.

Keywords: Software Router, NFV, DPDK

1. Introduction

Software Defined Network (SDN) and Network Function Virtualization (NFV) [1] are the paradigm shift for service providers and telecom companies, where they can provide and maintain their services 1) easily, 2) automatically, 3) swiftly, and 4) economically. Service providers and telecom companies are trying to attain a very flexible, controllable, and yet intelligent communication network, using software-based approach such as OpenFlow, OpenStack, and the virtualization technologies.

The users of these services also receive the benefits. For example, when Software Defined WAN (SD-WAN) services are established, the consumers will be able to set up a long haul private circuit in (say) thirty minutes. Similar benefits can be taken from the notion of Service Function Chaining (SFC) [2]. We could put a DDoS mitigation device in our virtualized network immediately, in the midnight, with just clicking the appropriate button with proper billing information. Previously such DDoS in the midnight could not be immediately dealt with, and setting up the new device in the network may take hundreds of thousands of dollars, and plenty of time such as weeks.

Toward the construction of such software-based flexible network, we tackle the challenge of high-speed software IP router. The performance of the network is the key: whatever the additional features are developed, the network is useless when the performance is low. The performance of software IP router, and further that of the virtualized version, is the key to the success of future network. Furthermore, if the high-speed packet processing performance that is required by the software-based IP router is achieved, then the technology that enables it can also be applied to other important network functions, such as firewalls and deep packet inspection (DPI) devices, improving their performance as well.

Slow performance has been worried for the software router for a long time of around twenty years. Recently a few promising technologies seemed to have overcome two important performance problems: the packet forwarding problem and the routing lookup problem. Intel DPDK [3] solves the problem of packet forwarding performance. Recently Poptrie [4] is invented and expected

to open a new horizon for high-speed software router by removing the IP routing lookup bottleneck.

In this paper we tackle the integration design of these promising technologies. We target to provide a robust, high-performance, reliable IP router software that can even be used as the basis of developing other important networking functions/middleboxes. Example of these important networking functions include the firewall, the Deep Packet Inspection (DPI), the Carrier Grade NAT (CGN), and the load-balancer.

Rest of this paper is organized as follows. Section 2 describes the related work, Section 3 explains the design principle and the software architecture, Section 4 exhibits the evaluation, and Section 5 the Conclusion. In the evaluation Section, provided are the performance benchmark (throughput, latency, jitter) of a real DPDK system, scaling of throughput by the use of multi-cores, the peak performance of our implementation, comparisons between locking/synchronization methods, and the performance of the overall integrated system.

2. Related Work

Several research on high-speed PC router has been conducted in the past, including Click modular router [5], and its recent variants as the platform for middlebox-VMs, ClickOS [6].

RouteBricks [7], and PacketShader [8] both claimed the usefulness of multi-queue, and both exhibited a superior performance. RouteBricks integrate multi-queues in Click, while the PacketShader added the GPU-accelerated IP routing lookup to the use of multi-queues.

Netmap [9] provides a programming framework that can efficiently share packet buffers or queues without redundant packet copy overhead. It essentially describes the technology similar to DPDK, with the exception that the Netmap provides a bit of memory/packet/process protection additionally.

Poptrie [4] provides a very fast IP lookup data structure. With this technology, it is expected that we do not need custom hardware devices, such as TCAM, GPU, and FPGA, even for the purpose of very high performance IP routing lookup.

We take the course of parallel processing to achieve the desired superior performance. Thus, we need some lock/synchronization framework. For the purpose, we use Read-Copy Update (RCU) [10]. Brocade's virtual router seems to take the same approach [11].

A past work [12] took almost the same approach with us. However, our paper differs from them in several points: we provide the overall router software design that supports other functionalities such as route change and display of running information. Also we provide more thorough evaluation including latency and jitter performance, packet loss ratio, and the comparison of RCU performance with other methods.

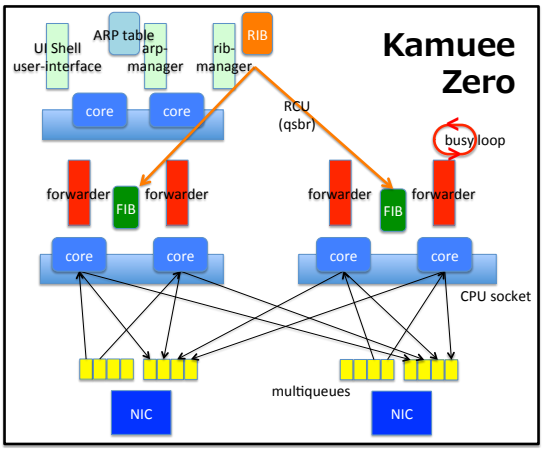


Figure 1: Internal structure

3. The Design of the Software Router "Kamuee"

Figure 1 depicts the internal structure of Kamuee. With the hope of future development of rich function versions, the current simple version is called Kamuee Zero (also written as Kamuee0). It is a simple DPDK-based application, with the design principle of the "designated thread" and the "internal messaging". By designated thread we mean that the data is basically managed by the single corresponding manager thread, avoiding the need for locks. Thus if a thread wants to access a data, it has to request the manager thread of the data, by the internal messaging. For instance, the routing table, which we call the Routing Information Base (RIB), is mainly managed only by the "rib-manager" thread. ARP manager thread (called "arp-manager"), and the UI-shell thread (called "user-interface") must request by the internal message to the rib-manager about the RIB contents. If the rib-manager changed the RIB contents, it constructs the forwarding table, called the Forwarding Information Base (FIB), that is actually used by the "forwarder" to forward packets. The FIB is published through the RCU process, hence the FIB access scales well with the number of reader process (i.e., the forwarders), along with the consideration of FIB update. We employed the Poptrie for the FIB data structure: the FIB is small enough to be loaded in the L3 cache of

CPU (e.g., less than 10 MB), and is shared among the CPU cores within the CPU.

Next-hop resolution during packet forwarding is achieved by accessing the FIB in the forwarder thread. The FIB is constructed based on the RIB and ARP table by the rib-manager thread. On update of either RIB or ARP table, the rib-manager thread constructs an updated version of the FIB and publishes it to the forwarder threads.

One way of implementing FIB publishing is by using traditional mutex locks. Mutex locks allows rib-manager thread to stop the forwarder threads temporarily to change the FIB table that is accessed from the old FIB to the new FIB. Mutex lock can be a global mutex to stop all the forwarder threads at once or it can be a per-thread mutex to stop forwarder threads one by one. However, using mutex lock to stop the forwarder thread temporary would mean that the router would stop forwarding packets temporary. This may cause packet loss, additional latency and performance drops, and therefore unacceptable in real world use cases.

In kamuee, to overcome the problem with mutex locking, a lock-free data synchronization mechanism called Read-Copy-Update (RCU) [13] is utilized. RCU keeps multiple versions of the data object on updates. When updating a data object, the old data object is swapped with the new data object and keeps the old data object intact. After all threads accessing the data object completes the data critical section, thus stops referencing the old data object, the data object is released. This mechanism enables the rib-manager thread to publish the new FIB without stopping the forwarder threads.

When using RCU, the algorithm for determining completion of the data critical sections have great effect to the packet forwarding performance of Kamuee. In Kamuee, the forwarder threads enter and exit the data critical section for each and every packet the forwarder thread handles. Therefore the overhead for determining the completion of the data critical section must be as low as possible.

The liburcu library [10], an open-source user-space RCU implementation, implements 3 algorithms for achieving determination of data critical section completion: 1. General-Purpose (MB), 2. Signal-Based (SIGNAL), 3. Quiescent-State Based Reclamation (QSBR). QSBR algorithm provides the lowest overhead to the read-side threads compared to other algorithms [10]. As having the lowest read-side overhead is critical to Kamuee's packet forwarding performance, Kamuee employs Quiescent-State-Based Reclamation (QSBR) algorithm to achieve determination of data critical section completion.

Table 1: The hardware and software that consists Kamuee0.

Kind	Product Name
M/B:	Supermicro X10DAX
Chassis	Supermicro SC836BA-R920B
CPU:	Intel Xeon E5-2687WV3 ×2
Memory:	DDR4-2133 16GB ×16 = 256GB
NIC:	Intel XL710-QDA1 ×4
OS:	Ubuntu 14.04.4
Data Plane:	Intel DPDK 16.04
Lookup:	Poptrie [4]

Table 2: Traffic Generator & Analyzer

Kind	Product Name
Traffic Generator Chassis	SPT-3U
Traffic Generator Module	MX-100G-F2
Tranceiver Adapter	ACC-6069A

4. Evaluation

The evaluation is done by using the Spirent TestCenter traffic generator, connected directly to the Device Under the Test (DUT) (i.e., our implementation). We provide current hardware configuration of our implementation and the test configuration, in Section 4.1.

We evaluate our implementation in several perspectives. First, in Section 4.3, we evaluate the basic forwarding performance that is provided thanks to DPDK. The basic forwarding performance benchmark exhibits 1) that our implementation does not impede the desirable high-performance of DPDK, and 2) a criterion for how much performance we can get at maximum in our test configuration, with some additional functions.

4.1. Test Configuration

Kamuee0 and the traffic generator consist of the hardware and software listed in Table 1 and 2, respectively. The Kamuee0, as the DUT, is connected to/from the traffic generator/analyzer, Spirent, directly with the four Direct Attach Copper (DAC, a.k.a., Twinax) cables through the 40Gbps Ethernet. Specifically, four 40GbE server adapters, XL710-QDA1, are installed in the Kamuee0, and each is connected via the 40GbE DAC cable. Hence the maximum amount of traffic that can be ingested to the Kamuee0 is 160Gbps in our test setup.

We generate test traffic load from the Spirent generator, ingesting it into the Kamuee0, and have Kamuee0 classify based on the destination address of the IP packet which port it should be emitted, and then the Spirent receives and counts the emitted packets.

The various test parameter settings are explained in the next section, Section 4.2.

4.2. Labels

40G/80G/160G Amount of traffic to support, and capacity of ports to be used; indicates the scale and the targets of the implementation and the test. It indicates, for example, how much traffic is ingested from the Spirent to the Kamuee0.

L3NONE/L3DEFAULTS/L3BGPFULL Routing table functionality that is employed in the test. L3NONE does not utilize the route table at all; the route is decided by the forwarder only by some bits in the IP destination address field (sometimes the first two bits), and then decided the forwarding port among the four interface ports.

L3DEFAULTS provide the route table that consists of only four route entries that collectively covers the whole IP address space. This is similar to the default route (0.0.0.0/0) which covers the whole space by just one route. The four default routes are 0.0.0.0/2, 64.0.0.0/2, 128.0.0.0/2, and 192.0.0.0/2. The nexthop of each route is destined to each port (port 1,2,3, and 4, respectively).

L3BGPFULL includes the route table setting that incorporate a BGP full-routes routing table, from the RouteViews project's archive, at LINX (the London IX) in the date of 2014-12-17, at peer 46th. The LINX route table includes 518,231 route entries. Additionally, L3BGPFULL includes the four default routes that is described in the L3DEFAULTS above. This is because if we lack these routes the packet will be dropped when the route was not found in the BGP full-routes table. In order to distribute the traffic among the four interface ports, the above L3DEFAULTS routes are incorporated additionally to the BGP full-routes table, so that maximum performance can be measured including the route-not-found cases.

Q1-Q4/Q4m-Q6 The number of queues per port that are used to forward traffic. We used a CPU core per each queue, so the number of queues is equivalent to the number of CPU cores used. "Q*n*" means a setup where *n*-queues are used per port to forward traffic. We use up to 4 queues per core, i.e., Q1-Q4, in most cases, but in some specific cases we also evaluate the case of 5 and 6 queues per core (Q5 and Q6). We could not allocate four core/queues to each four interface port, because we were short in the number of cores, given that only the 15 cores are available for the use in DPDK, and 5 cores are reserved for other functions such as OS, rib-manager, and UI-shell. Hence the last 4th port is assigned only 3 core/queues. This setting is indicated by "Q4m".

BACK/STRAIGHT/CROSS Indicates the path of the test traffic. BACK means the traffic is returned back to the received port. This situation is not likely to happen in the routers in the real field. STRAIGHT means the traffic is forwarded in the other NIC port of the same CPU. CROSS means that the test traffic is destined to the NIC port that is connected to other CPUs.

WRONGNUMA The core from a differnt CPU is intentionally assigned to a NIC port. This means it will incur redundant traffic and distance otherwise not needed. WRONGNUMA is used to check the overhead of using the distance CPU cores.

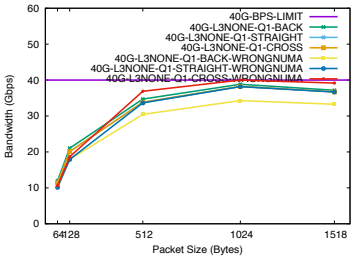
Unless otherwise noted, the QSBR-mode RCU is employed for the synchronizing mechanism, throughout the evaluations in this paper.

4.3. Single Core/Queue Forwarding Throughput

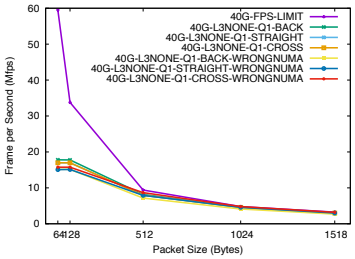
Figure 2 shows the performance benchmark using only the single queue/core setting. Note that there is no routing lookup involved in the process. This test should be equivalent to the native DPDK performance test.

BACK/STRAIGHT/CROSS means the path of the test traffic: BACK means that the test traffic is returned back by the Kamuee0 router (the Device-Under-Test: DUT) on the same interface port. This configuration is just informational, because in the real network, putting back the traffic to the direction where it came is meaningless, and almost will not happen. STRAIGHT means that the traffic is forwarded to the other interface port within the same CPU (Note that the PCI-e slot is connected below each CPU). On the other hand, if the traffic go across the CPUs, then we call it the CROSS traffic, and the rather increased latency or throughput limitation is expected, because the traffic is supposed to go through the bus between the CPUs (e.g., QPI).

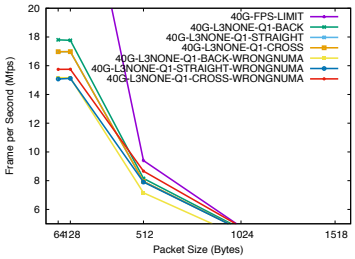
First, from Figure 2b we see that we cannot achieve 40GbE wire-rate performance if we use only a single queue per port. Note that all settings in the Figure 2b employ only a single queue/core per port. When the size of the packet grow such as 1024 bytes, it is close to the wire-rate, but it still does not fill the bandwidth fully. This lack of fulfillment of performance requirements by a single busy-loop packet processing shows the necessity for the multi-core parallel processing design. It supports our design principle of multi-core design for high-speed packet processing.



(a) Bits per Second



(b) Frames per Second



(c) Frames per Second (Zoom of Figure 2b)

Figure 2: Single queue/core throughput

Secondly, the NUMA architecture and its configuration impacted the performance, but not as much as was expected (or afraid). Notice that even in the case of WRONGNUMA configuration where the CPU core is used from the opposite side of the CPU, the performance degradation is marginal.

A little bit interesting phenomenon can be seen in the result. The rare configuration of CROSS-WRONGNUMA shows a rather better performance compared to others, somewhat surprisingly. The WRONGNUMA label means that the CPU core (and hence the memory) was assigned from the opposite side CPU socket, from the standpoint of the receiving interface port. Also, the CROSS traffic means that the test traffic is destined to the interface port below the other (opposite side) CPU. Hence, in this case it means that the packet reception is done by the distant CPU core/memory from the NIC, and in the transmission part the CPU core/memory is closer to the NIC on which the packet will be transmitted. From the result, we can interpret that it is better for us to assign to the packet a CPU core that is close to the transmission NIC, rather than reception NIC.

This is attributed to the DMA-read/write overhead. In the packet reception process the NIC is writing to the CPU's memory (DMA-write), and in the packet transmission process the NIC is reading from the CPU's memory (DMA-read). In general, DMA-read has larger cost than DMA-write, because it incurs multiple steps involving the reading of descriptor table.

This slightly better performance of CROSS-WRONGNUMA can also be seen in the frame per second (fps) performance (Figure 2c), and in the later performance comparison of average latency and jitter.

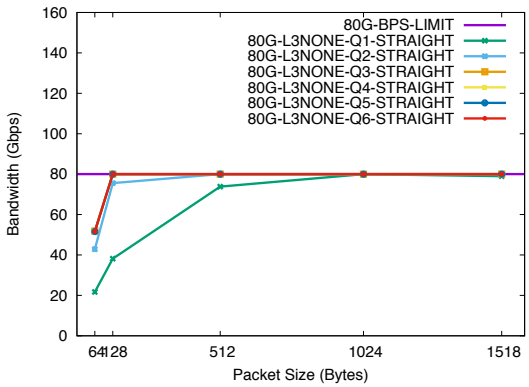
From these results, we can say it has more impact that the assigned CPU core(memory) is closer to the transmitting NIC port, rather than to the receiving NIC port. In other words, it has only a little meaning to carefully design the packet handling CPU cores assigned closer to the receiving NIC port. Given that until the completion of routing lookup we cannot know apriori which NIC port the packet will be forwarded, we cannot assign in advance the CPU core that is closer to the transmission NIC port. A design choice of disregarding the CPU/memory distance in NUMA architecture might seem effective.[1] As a router's design choice, sometimes it may be desireable to assign the CPU cores even from different (distant) CPU sockets, because the NUMA distance impact may be marginal, while the contribution of additional CPU core is not.

4.4. Scaling Throughput

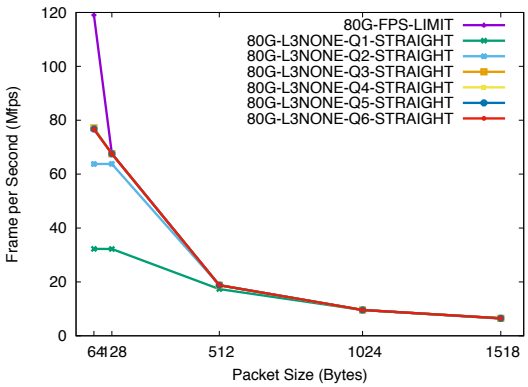
One of the prominent feature of Kamuee is the flexible configuration of core/queue placement, which enables easy scaling of performance using multiple cores and queues. Figure 3 illustrates the scaling of multi-core/queue to support unidirectional 80Gbps traffic (unidirectional 40GbE x 2). We made a rather strong assumption on the traffic flow direction, and assigned almost all core/queue pairs to the receiving side interface port (i.e., the router performance is not symmetric). It shows that by using more than three cores/queue pairs (labeled as Q3), Kamuee supports almost full 40GbE x 2 wire-rate traffic in 128B packet size or larger. Wire-rate traffic of smaller-than-128B packet sizes are limited by the limitation of the Ethernet NIC hardware in use. The fact that we can achieve the wire-rate with only three core/queue pairs means the room for higher performance, and it also enables us to conduct other services in the high-performance packet forwarding process, such as routing lookups and packet classifications.

Figure 4 shows the maximum input/output throughput for our configuration: 40GbE x 4 = 160Gbps. We

[1] As WRONGNUMA label name suggests, the authors of this paper did not think like that previously.

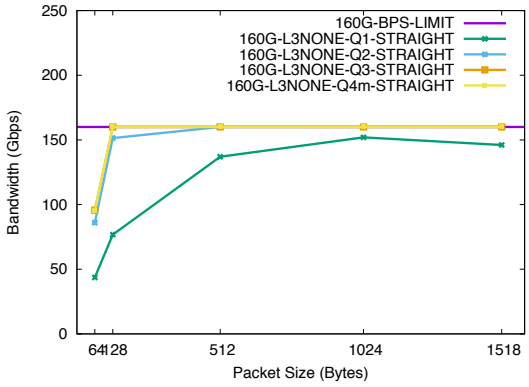


(a) Bits per Second

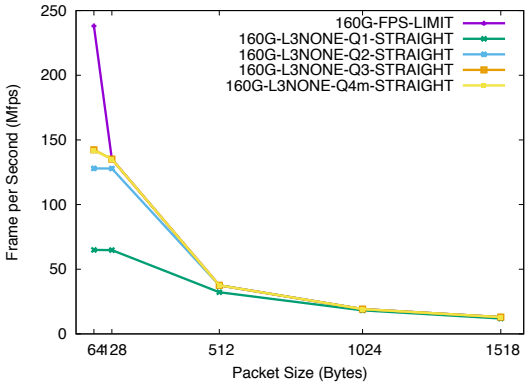


(b) Frames per Second

Figure 3: 80Gbps Throughput



(a) Bits per Second



(b) Frames per Second

Figure 4: 160Gbps Throughput

could only use 15 cores for packet forwarding (other 5 cores are used by OS, rib-manager, arp-manager, UI-shell, and master thread), so the port 3 only had 3 cores while port 0 to 2 had 4 cores (hence the label Q4m). Our router showed 160Gbps of forwarding capability on the 128B short packet traffic. On this router, we will add routing lookup feature later in this paper.

4.5. Latency and Jitter

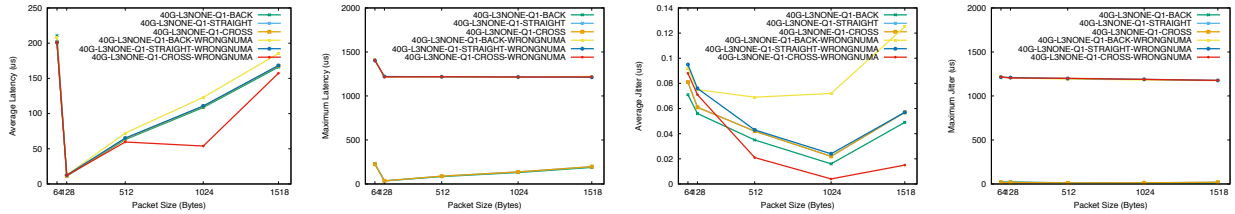
Figure 5 shows the latency and jitter of approximately ten seconds duration. They employ only a single queue/core, do not include routing lookup, and are compared against the traffic path (BACK/STRAIGHT/CROSS), and NUMA distance.

We infer that, since the Maximum Read Request Size register of the PCI Express was set to 128 bytes, the latency was smallest when the Ethernet frame size is 128 bytes.

For the maximum latency and maximum jitter, the result showed the clear distinction of WRONGNUMA from others. All WRONGNUMA configuration did have the maximum latency or jitter more than 1000us (1ms). We suspect that this is attributed to the DMA write process to a distant CPU core that involves a largest time. Overall, latency and jitter are kept well low, unless it is WRONGNUMA.

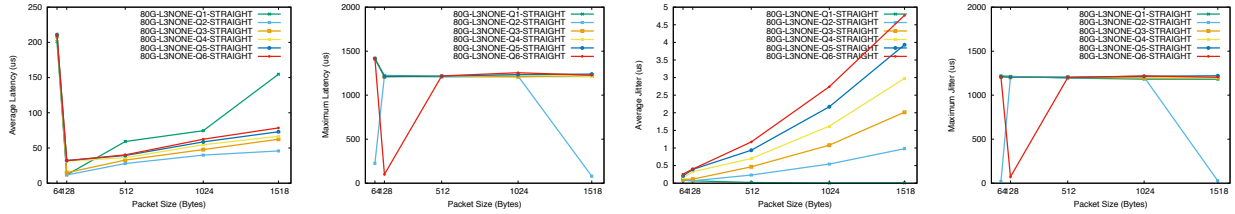
As stated in the previous section, CROSS-WRONGNUMA exhibits a little bit interesting behavior. CROSS-WRONGNUMA shows a rather better performance in average latency and average jitter. However, it incurs the large maximum latency and jitter, just like the other WRONGNUMA configuration.

Figure 6 illustrates the relationship between the latency, the jitter, and the number of queue per port. Figure 5 uses only a single queue, and the maximum latency/jitter were kept quite low (unless WRONGNUMA). In contrast, the Figure 6 showed that



(a) Average Latency (b) Maximum Latency (c) Average Jitter (d) Maximum Jitter

Figure 5: 40G: Latency and Jitter



(a) Average Latency (b) Maximum Latency (c) Average Jitter (d) Maximum Jitter

Figure 6: 80G: Latency and Jitter

almost always the maximum was larger than 1000us (1ms). We suspect some kind of contention is occurring in the PCIe process. From this result the increased number of core may deteriorate the maximum (worst-case) latency and jitter.

4.6. Packet Loss

Packet loss by current Kamuee is quite large. Figure 7a and 7b shows the packet loss of Kamuee software router for 1 queue on 40Gbps traffic, and 1 to 6 queues on 80Gbps traffic, respectively. Within the only approximately 10 seconds of test, more than a million packets (sometimes hundreds of millions packets) have been lost in the single 40G test, as shown in Figure 7a. Since the 1 queue cannot handle 40Gbps traffic by itself, the lack of performance result in the amount of packet loss. The number of packet loss does not seem to be significantly different for the traffic path inside the router (BACK/STRAIGHT/CROSS).

The packet loss derived from the shortage of packet handling throughput can be removed by adding the multiple queue/core. From Q3 and larger number of queues in Figure 7b, the number of lost packets decreased from 1×10^8 to 1×10^5. However, after the removal of the shortage of packet handling throughput, the packet loss does not improve further even when the queue/core pairs are added. Specifically, increasing from Q4 to Q6 does not exhibit a significant improvement.

In order to show the relationship between the amount of load and the number of packet loss, we conducted the packet loss measurements with the ranging load from 20Mbps to 1Gbps by the steps of 20Mbps, for 60 seconds duration each. Figure 7c shows the relationship, and we can see that even in the very low load such as less than 400Mbps, the packet loss occurs.

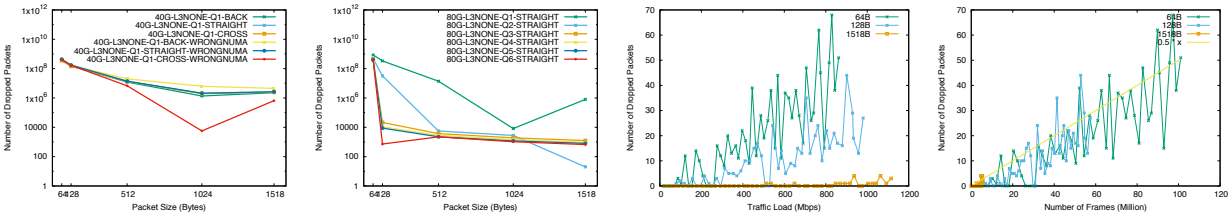
Figure 7d shows that the packet loss does not depend on the packet size: the number of packet losses occurred per 10 million packets does not seem to differ significantly between 64B and 128B traffic, at least for the range from 0 to 60 million packets.

4.7. RCU and Locks

Figure 8a and Figure 8b presents the throughput of the Kamuee0 when different FIB synchronization methods are used. Traffic of 80Gbps was generated by Spirent to test the forwarding performance of the Kamuee0. The kamuee0 ingested the traffic from 2 out of 4 ports and emitted the traffic to the 2 ports not receiving traffic. 4 forwarder threads were running per port and therefore, actively accessing the FIB when handling traffic. The L3BGPFULL configuration of the FIB was used during the measurement[2].

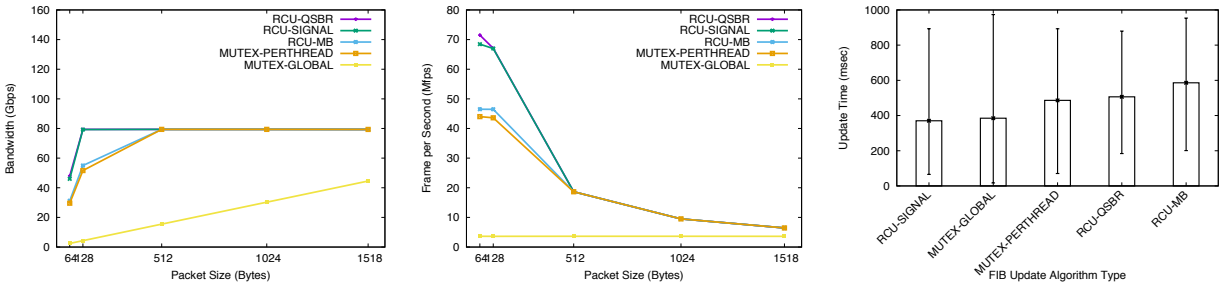
RCU with QSBR algorithm provides the best bandwidth throughput and frames per second throughput when handling 64 byte packets. When handling 128

[2]The L3BGPFULL configuration used in this test did not include the default routes mentioned previously. Instead, the packet was forwarded randomly to either ports when route was not found.



(a) 40G packet loss (b) 80G packet loss (c) Packet Loss against Traffic Load (d) Packet Loss against Number of Frames

Figure 7: Packet Loss



(a) Bits per Second (b) Frames per Second (c) Time required to update and synchronize FIB

Figure 8: Comparison of throughput with each data locking algorithm

byte packets, the performance of RCU with SIGNAL algorithm and RCU with QSBR algorithm provides the best performance. With packet sizes over 512 bytes, all methods except for global mutex lock provides equal performance. Overall, RCU with QSBR algorithm provides the best performance.

RCU requires memory barriers to achieve lock-free data synchronization. The paper on user-space RCU implementation [10] discusses the problems with memory barriers and its impact to the performance. Basic concept of RCU is that there is a global counter that keeps the history of data change and each read-side threads keep track of the referencing data version, which is a snapshot of the global counter, locally. When updating the a global reference counter, the reader threads must execute a memory barrier in order to ensure that read-side threads access the global counter after the counter has been properly updated.

The difference in 3 RCU algorithms is the strategy on how to use the counter and when to execute memory barrier. MB algorithm is designed to be a general-purpose RCU and provide flexibility. Therefore, with MB algorithm, memory barrier is executed every time a read-side thread go in to the data critical section. With Kamuee0, the data critical section contains FIB lookups which need to include a memory barrier, and may cause the performance to drop. QSBR and SIGNAL algorithms minimizes the use of memory barriers to provide higher read-side performance than the MB algorithm. SIGNAL algorithm sends POSIX signals from the write-side thread when memory barriers are needed. The read-side threads only execute memory barriers when signal is received. With QSBR algorithm, a read-side thread periodically makes a snapshot of the current global counter and a memory barrier is only executed at periodic snapshot. SIGNAL algorithm requires forwarder threads to mark entries and exits of data critical section while QSBR algorithm only requires a periodic snapshot of the global counter. Therefore, though both QSBR and SIGNAL algorithm minimizes the use of memory barriers, QSBR has slightly better read-side performance.

Figure 8c presents the time required to synchronize a new FIB between forwarders with different synchronization methods. To observe the synchronization time during the maximum load, 80Gbps of 64 byte packets were used to put stress on the Kamuee0.

RCU with SIGNAL algorithm provided the fastest synchronization time, followed by global mutex lock, per-thread mutex lock, RCU with QSBR algorithm and

RCU with MB algorithm. Kamuee0 employs RCU with QSBR algorithm, although it does not provide the best update performance. RCU with QSBR algorithm was employed as the packet forwarding performance is more critical to the design of Kamuee0.

4.8. Overall Performance

Figure 9 illustrates overall peak performance of our current best setup. Collectively the 160Gbps traffic is ingested from all the four interfaces. The throughput is shown in Figure 9a. The difference in throughput between L3NONE and L3DEFAULTS shows the overhead of this routing lookup framework; L3DEFAULTS employs the routing lookup framework but the number of route table entries are just four. The difference between L3DEFAULTS and the L3BGPFULL are the impact of the size of the route table. We see the both overhead is significantly small. The L3BGPFULL (i.e., the Kamuee0 with BGP full-routes) exhibited 145Gbps throughput in the routing of 128B short packet traffic.

Note that in this evaluation method, we cannot distinguish the loss of traffic due to the overhead of routing lookup, from the loss of traffic caused just by the skewed balance of traffic between interface ports. BGP full-route table is NOT completely balanced in the size of the coverage by each routing table entry. Hence like our setting if each route entry is equally assigned to each port, then there will be some skewed balance in traffic; some ports get more traffic, some less. Since the ports that get more than 40Gbps will drop some packets, it is possible that there are packet losses even if the performance of the router is not a problem. The decreased amount of forwarded traffic of 145Gbps, against full 160Gbps, can also be attributed to this skewed balance in the BGP full-route table.

The latency and jitter of the same setting are shown in Figure 9b and 9c. Latency is increased due to the increased size of the route table (i.e., comparison between L3DEFAULTS and L3BGPFULL). Jitters can be as large as more than 1 millisecond.

Table 3 shows the packet loss count and ratio for the duration of 60 seconds in this configuration. The packet loss is significantly increased from L3DEFAULTS to L3BGPFULL, both for 128B and 1518B sized packets.

5. Conclusion

We have shown the design and implementation of a high-performance software router, called Kamuee0. Its simple design enables the extensibility for performance scale, and for additional features.

It exhibited as much as 145Gbps throughput in the 128B short-packets traffic. The performance in latency, jitter, and packet loss still have some rooms to improve.

Acknowledgements

We appreciate Toyo Corporation for they lent us a hand and the device necessary to conduct our evaluation. We would like to thank also to Masafumi Oe, Hirochika Asai, and Takeshi Matsuya for their valuable technical comments and advices.

References

[1] M. Chiosi, et al., Network Functions Virtualisation – An Introduction, Benefits, Enablers, Challenges & Call for Action, `https://portal.etsi.org/NFV/NFV_White_Paper.pdf` (2012).

[2] J. Halpern, C. Pignataro, Service Function Chaining (SFC) Architecture, RFC 7665 (Informational) (Oct. 2015).
URL `http://www.ietf.org/rfc/rfc7665.txt`

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

[6] 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.

[7] 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.

[8] S. Han, K. Jang, K. Park, S. Moon, Packetshader: A gpu-accelerated software router, in: Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM '10, 2010.

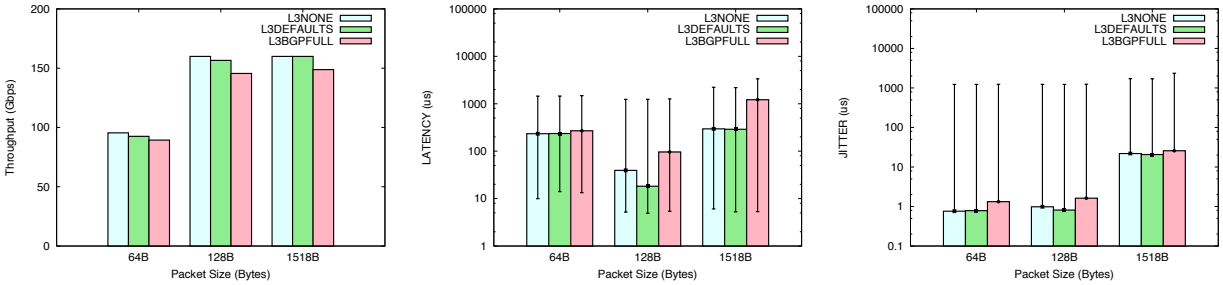
[9] L. Rizzo, netmap: A novel framework for fast packet i/o, in: 2012 USENIX Annual Technical Conference (USENIX ATC 12), 2012.

[10] M. Desnoyers, P. E. McKenney, A. Stern, M. R. Dagenais, J. Walpole, User-level implementations of read-copy update, IEEE Transactions on Parallel and Distributed Systems 23 (2012) 375–382.

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

[12] 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.

[13] 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.



(a) Throughput (b) Latency (c) Jitter

Figure 9: Overall Performance

Table 3: Overall Packet Losses

	64B			128B			1518B		
	#pkts	#dropped	(%)	#pkts	#dropped	(%)	#pkts	#dropped	(%)
L3NONE	17,888,594,284	7,216,753,509	(40.34%)	10,170,006,489	2,321,501	(0.02%)	987,932,796	252,610	(0.03%)
L3DEFAULTS	18,042,666,673	7,606,593,667	(42.16%)	10,159,437,842	215,961,305	(2.13%)	977,434,801	250,711	(0.03%)
L3BGPFULL	18,029,988,569	7,961,881,397	(44.16%)	10,157,535,139	918,556,928	(9.04%)	979,805,048	68,834,701	(7.03%)