A High-Performance OpenFlow Software Switch

Reza Rahimi & M. Veeraraghavan
University of Virginia
Charlottesville, VA, USA

Y. Nakajima & H. Takahashi
NTT Labs
Tokyo, Japan

Y. Nakajima, S. Okamoto & N. Yamanaka
Keio University
Yokohama, Kanagawa, Japan

***Abstract*—Software switches offer flexibility to service providers but potentially suffer from low performance. A software switch called Lagopus was implemented using Intel's Data Plane Development Kit (DPDK), which offers libraries for high-performance packet handling. Prior work on software switches focused on characterizing packet forwarding throughput. In this work, we evaluated the impact of certain parameters and settings in Lagopus on application performance and studied packet drop rates. The importance of receive-thread packet classification for load balancing and to send delay-sensitive flows to a different worker thread from high-throughput flows was first demonstrated. Next, we showed that a loop-count variable used to control packet batching should be kept small in case link utilization is low. Finally, we showed that packet drop rate could be non-zero when the OpenFlow table size is large and packet arrival rate is high, and interestingly, the packet drop rate was higher with four worker threads than with a single worker thread. This implies a need for careful calibration and planning of the parameters of parallelization.**

Keywords: SDN, NFV, software switch, OpenFlow, DPDK

I. Introduction

Packet forwarding is typically implemented in hardware to handle complex operations such as longest-prefix match of destination IP addresses or high-rate lookups of MPLS labels in service provider networks. In OpenFlow based Software Defined Networks (SDN), the matching operation in table lookups can be even more complex if multiple header fields are used. Hence, service providers are likely to need hardware based switches that can perform high-rate lookup operations.

However, with hardware switches, even in the SDN paradigm, service providers are dependent on switch/router vendors to upgrade their products in support of new features. Software switches/routers are thus attractive to service providers as they offer more flexibility. The drawback of software switches is lower performance if standard Linux kernel based packet processing methods are used. To improve performance, several new packet I/O solutions have been developed on top of which packet-forwarding and other network functions can be implemented.

This approach of implementing data-plane functions in software is called Network Functions Virtualization (NFV) [1], [2]. The term Virtualized Network Function (VNF) is used to describe a software implementation of a particular network function such as packet forwarding and packet filtering.

The objective of this work is to evaluate a high-performance software switch called Lagopus [3] that was designed for WAN service providers. Lagopus is designed for multi-core execution, and can run a configurable number of receive I/O threads, worker threads for flow table lookup operations, and transmit I/O threads. We executed Lagopus on two different testbeds, and characterized not only packet forwarding rates but also the impact of Lagopus parameter settings on application performance. Two types of applications were run: throughput-sensitive bulk-data flows, and delay-sensitive flows. Our work also tests the capability of Lagopus to handle packet forwarding at 10 Gbps when run on an off-the-shelf multi-core commodity PC.

Our key findings are as follows: (i) receive thread should be configurable to perform sophisticated flow matching for both load balancing and for directing delay-sensitive flows to a different worker thread from high-throughput flows; (ii) a small loop count should be used in packet batching to keep packet delays small when link utilization is low; (iii) at high packet arrival rates (10 Gbps), while it may appear that packet handling rates are close to link rates (10 Gbps), packet drops were observed. Interestingly, the packet drop rate increased to 3% with four worker threads from 0.27% with a single worker thread because the average per-packet processing delay was higher than with one worker thread. As packet losses can cause significant TCP-flow throughput drops on high bandwidth-delay product paths, software switches designed for high-rate packet forwarding should be calibrated for zero drop-rate performance, and the number of threads used should be selected carefully.

Section II provides a brief review of related work. Section III describes Lagopus. Section IV describes our experiments for studying the impact of certain parameters on performance. The paper is summarized and concluded in Section V.

II. Related work

Virtual switches first emerged to support packet switching between virtual machines within a physical machine, but now some of these virtual switches are also used as software switches. For example, Open Virtual Switch (OVS) [4], [5] can be used to provide packet switching functionality between VMs within a physical machine, or as a software switch that provides packet forwarding between multiple physical Network Interface Cards (NICs) of a dedicated server.

Given the potential benefits, a number of projects have implemented software switches. A 2011 paper describes a solution called RouteBricks [6] in which a cluster of 4 Nehalem servers interconnected by a full mesh with 10 Gbps external

978-1-4799-8950-8/16/$31.00 ©2016 IEEE

links could support 64B packet forwarding at 12 Gbps, which means each server could support an external link at 3 Gbps. Bianco et al. [7] compared the performance of L2, L3 and OpenFlow switching within the Linux kernel. Packets were generated by an Agilent N2X router tester [8] and sent on 1 Gbps links to a Linux host. With 1500B packets, forwarding capability close to 1 Gbps was reported.

Pongrácz et al. [9] reported forwarding rates between 5.26 Gbps and 9.6 Gbps for packet sizes of 64B and 512B, respectively, with one OpenFlow 1.3 rule. This implementation was based on Intel's Data Plane Development Kit (DPDK) [10], [11], which was designed specifically for high-throughput packet processing. Rizzo et al. [12] developed a new packet I/O framework called `netmap`, and demonstrated the use of this framework by running OVS and Click [13] on `netmap`. A value of 10,600 Kpps was reported for OVS performance with 64B packets, which amounts to 5.5 Gbps. A discussion of the pros and cons of high-performance packet IO implementations, specifically, `netmap`, PF_RING ZC, and DPDK, was presented by Gallenmüller et al. [14]. Emmerich et al. [15] compared OVS, IP forwarding, Linux bridge and DPDK vSwitch [16], which is a modified version of OVS that leverages high-performance capabilities of DPDK. With a single core of a 3.3 GHz Intel Xeon E3-1230 V2 CPU, the packet-forwarding throughput between physical NICs was 11.31 Mpps for DPDK vSwitch in contrast to 1.88 Mpps for OVS, 1.58 Mpps for IP forwarding, and 1.11 Mpps for Linux bridge.

III. Lagopus: a software switch

Lagopus is a software OpenFlow switch designed for wide-area network service providers [17]. Lagopus is designed to be run on multi-core processors. Lagopus leverages Intel's DPDK drivers and libraries.

Packet processing at a switch consists of: (a) packet reception, (b) packet header parsing, (c) packet classification, (d) flow-table lookup to determine outgoing port, (e) QoS mechanisms such as policing on the ingress and scheduling on the egress, and (f) packet transmission. Most of these operations require memory accesses, and memory access rates are currently lower than CPU data-processing rates. Therefore, software switch designs should aim to minimize data movement between NIC buffers, main memory, and processor caches. Further, while using multiple cores for high packet processing rates, designs should place data intelligently given the non-uniform nature of memory accesses in multi-core processors.

A. Lagopus architecture

Fig. 1 shows an example Lagopus architecture in which eight cores are used, two cores for I/O receive threads, two cores for I/O transmit threads, and four cores for flow-table lookup worker threads (referred to as "worker threads" in the rest of the paper). The receive and transmit threads use DPDK libraries and drivers to move packets in and out of NICs. Ring buffers are shared between the I/O receive thread and the worker thread on the ingress side, and between the worker thread and the I/O transmit thread on the egress side.

Lagopus exploits multiple cores using parallelization rather than pipelining. In other words, all functions executed on a single packet header are handled by the same worker thread, while different packets are sent to different worker threads.

To achieve high performance, the Lagopus design uses the following Intel DPDK features: (i) polling mode instead of interrupts to move packets received by NICs, (ii) lockless queues with ring buffers to provide multiple readers and writers simultaneous access to shared data, (iii) huge pages to reduce the rate of Translation Lookaside Buffer (TLB) misses, and (iv) Intel's Data Direct I/O technology [18].

Further, the Lagopus implementation includes: (i) packet classification for load balancing with explicit assignment of packets to worker threads, (ii) batch handling of packets (packet coalescing) between the NIC and the application, and (iii) high-performance flow-table lookup solutions [19].

B. Use of DPDK features

Polling is supported by DPDK. Polling is typically better when the packet hit rate is high. When compared to interrupt mode, polling is better for achieving lower response times. In a standard Linux implementation, interrupt mode is used to handle received packets since hosts perform other tasks besides packet handling. On the other hand, since a Lagopus host is dedicated to packet processing, to avoid the overhead of interrupt servicing, DPDK's polling feature is used. A negative aspect of polling is higher power consumption since the CPU core that executes the receive thread is utilized at 100% in a tight polling loop.

Lock-free queues are an important mutual exclusion solution necessary for high-performance systems [20]. As the I/O receive thread writes packets to the ring buffers shown in Fig. 1, the worker threads read packet headers to perform their tasks. A similar need arises on the transmit side.

The use of huge pages to reduce the rate of TLB misses is important in packet processing given that most tasks involve memory accesses.

Finally, the data direct I/O technology provided by DPDK moves data directly from the NIC to the last-level processor cache, and thus reduces main memory accesses.

C. Packet classification for load balancing

The load balancing approach implemented in Lagopus 0.2.2 divides packets between the multiple worker threads by first classifying packets by their source and destination MAC addresses and the next four bytes (which form the 802.1Q header for VLAN tagged Ethernet frames). In this implementation, all packets of a TCP flow are sent to the same worker thread, which is important for sequenced packet delivery. Out-of-sequence packets in a TCP flow could cause the fast-retransmit/fast-recovery procedure to execute, which in turn could result in reduced throughput.

DPDK includes functions that implement more sophisticated packet-classification algorithms. In a future release of

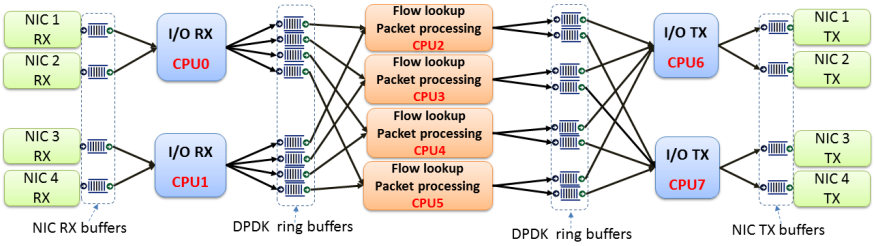

Figure 1. Lagopus architecture; I/O RX: Receive thread; Flow lookup packet processing: Worker thread; I/O TX: Transmit thread

Lagopus, these DPDK functions will be used to offer providers more configuration choices for how flows are distributed to different worker threads.

D. Batch processing of packets

Packet batching is a useful tool to lower overhead and improve packet handling rates. On the ingress side, packets are bunched together in the incoming NIC before being moved to ring buffers (see Fig. 1). On the egress side, worker threads bunch together packets before moving them to the outgoing NIC.

There are two controlling parameters for packet batching: (i) a burst size vector, and (ii) two loop counts. The burst size vector is denoted {(A, B), (C, D), (E, F)}, where A: I/O receive thread read burst size from the receiving NIC, B: I/O receive thread write burst size to the input ring buffers, C: worker thread read burst size from the input ring buffers, D: worker thread write burst size to the output ring buffers, E: I/O transmit thread read burst size from the output ring buffers, and F: I/O transmit thread write burst size to the transmitting NIC.

The loop counts specify the maximum number of times the I/O receive thread, worker thread, or I/O transmit thread, will execute their corresponding functions before pushing packets to the next set of buffers. If the packets of the specified burst size accumulate before the loop completes, the packets will be pushed to the next buffer. However, when utilization is low, these loop counts are useful to limit packet delays. The loop count is the set by the same variable (IO_FLUSH) for the I/O receive and transmit threads, while a different variable (Worker_FLUSH) controls the worker-thread loop count.

Thus, packet coalescing is used in multiple interfaces between the NIC, I/O receive thread, worker thread and I/O transmit thread.

E. High-performance flow-table lookup

For flow-table lookup, Lagopus 0.2.2 uses an OpenFlow-aware modified Patricia trie [3]. Patricia tries are good for exact-match lookups, but not well suited for longest-prefix matching [19]. Future releases of Lagopus will support alternative flow-table lookup algorithms suited for longest-prefix matching necessary for Layer-3 (IP) packet forwarding.

Lagopus also uses a flow cache to store frequently accessed flow-table entries. A flow-cache entry could be a combination of entries from multiple flow tables. Significant speed-ups in packet processing are possible with a flow cache [17].

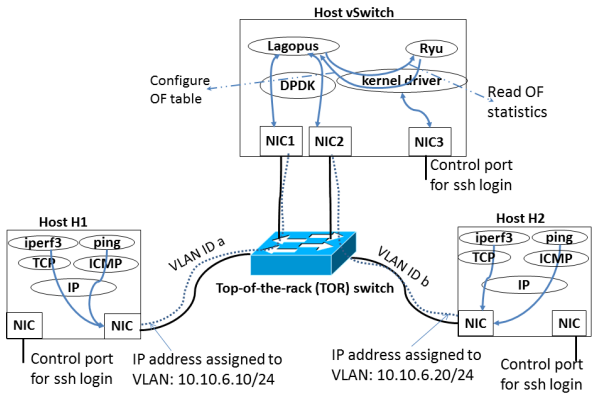

Figure 2. Experimental InstaGENI setup; OF: OpenFlow

IV. Experiments

Section IV-A describes the experimental testbed. Section IV-B describes the input parameter values used in our experiments, output measures, and the experimental workflow. Section IV-C describes a set of experiments to study the effects of packet batching on delay under low utilization. Section IV-D presents experimental results quantifying the impact of parallelization by varying the number of worker threads on applications. Section IV-E describes a high-throughput experiment to study the effects of OpenFlow table size.

A. Experimental setup

We used two experimental setups: (i) InstaGENI setup [21], and (ii) Keio University ("Keio") setup.

The InstaGENI setup consisted of a slice of three bare-metal (non-virtualized) hosts located in a University of Utah rack. All the hosts used in this experiment have two 8-core CPUs (Intel Xeon CPU E5-2450) and 24 GB RAM. Each host has four 1 Gbps Ethernet (GE) NICs, one of which is used as a "control port" for remote user logins and administrator access, and two more were used in our experiments. Ubuntu 14.04 with kernel version 3.13 was run on all three hosts.

Table I
INPUT PARAMETERS

Parameter	Value
Number of receive threads	1
Number of transmit threads	1
Number of worker threads	$1, \cdots, 4$
I/O thread loop count	{100, 10000} packets
Worker thread loop count	1000 packets
Burst size (InstaGENI)	144 packets
Burst size (Keio)	32 packets
OpenFlow table size	$3, \cdots 1M$ entries
Statistics collection interval	10 seconds

Fig. 2 illustrates the InstaGENI setup consisting of hosts H1 and H2 that were used for the application flows (`iperf3` [22] to generate high-throughput flows, and `ping` to emulate delay-sensitive flows), and host vSwitch that was used to run Lagopus and an SDN controller called Ryu [23]. The specific version used in all our experiments was Lagopus 0.2.2. Ryu configured the OpenFlow tables used by Lagopus for packet forwarding, and collected OpenFlow statistics from Lagopus.

Virtual LANs `VLAN a` and `VLAN b` were configured from host H1 and host H2 to the vSwitch host, respectively, both passing through the top-of-the-rack (TOR) switch. A rate of 1 Gbps was specified for these VLANs. The InstaGENI aggregate manager set up these VLANs by configuring the TOR switch and the hosts. The IP addresses assigned to the VLANs at hosts H1 and H2 were in the same subnet as shown in Fig. 2, and the vSwitch host VLANs was not assigned IP addresses. When running DPDK, the kernel Ethernet driver was disabled in host vSwitch as shown in Fig. 2. For Ryu-Lagopus communications, TCP/IP sockets were used over the kernel driver.

The Keio setup consists of two hosts. Host 1 has one 4-core Intel(R) Xeon(R) E5345 2.33GHz CPU and 8 GB RAM, while Host 2 has two 10-core Intel® Xeon® E5-2660 v3 2.60 GHz CPUs and 64 GB RAM. Host 1 has a Myricom Myri10GE dual-port NIC, while Host 2 has one Intel® 82599EB 10-GE SFI/SFP+ NIC. The hosts were directly connected to each other. A packet generator called `pktgen` [24] was executed on Host 1 to generate MPLS packets at 10 Gbps. An OpenFlow table was configured in the second host via Ryu to execute MPLS label matching. Lagopus was executed on Host 2. It received MPLS packets and transmitted all packets out on to the same link as there was only one link between the two hosts.

B. Experimental execution

This section describes input parameters, output metrics, and the experimental workflows.

Input parameters

Table I lists the input parameters with corresponding values. As shown in Fig. 1, there can be multiple receive I/O threads, multiple transmit I/O threads, and multiple worker threads. The number of threads of each type used in our experiment are listed in the first three rows of Table I. Row four shows that we experimented with two values of the I/O thread loop count. The next row shows that all experiments used a fixed worker thread loop count.

All six components of the burst-size vector were set to the default value of 144 packets in the InstaGENI setup, but per DPDK instructions [25] for the type of NICs used in the Keio setup, we reduced all six burst-size components to 32. With the default 144-packet setting, the packet drop rate in the Keio setup was high, yielding a packet forwarding rate of only 2 Gbps. When the burst size components were decreased to 32, a packet forwarding rate of 10 Gbps was measured.

Since our experiments study the impact of the ingress polling-loop count, OpenFlow table size, and number of worker threads, multiple values are indicated for these parameters in Table I. The last row of the table shows the intervals at which Ryu issues a request to Lagopus to read OpenFlow statistics.

Output measures For the InstaGENI testbed experiments, we used two output measures: `ping` delay, `iperf3` throughput, and for the Keio testbed experiment, we measured packet forwarding rates and packet drop rates. `Ping` delay and `iperf3` throughput are reported in logs generated by the corresponding applications at the end hosts, while forwarding and drop rates were computed from the byte and packet counts collected by Lagopus and obtained by Ryu.

Experimental workflow First, a Ryu script was executed to set up the OpenFlow table. The default setting of the OpenFlow table had only 3 entries: (i) an entry for handling flows in the H1 $\rightarrow$ H2 direction, (ii) an entry for handling flows in the H2 $\rightarrow$ H1 direction, and (iii) an entry for sending unmatched packets to the SDN controller. The matching field was input VLAN ID. The action specified for the first two entries was to modify the VLAN ID before sending the packet on the output interface. For example, for an application running on hosts H1 and H2, VLAN tags on packets received by Lagopus on NIC 1 of host vSwitch are modified from `VLAN a` to `VLAN b` before transmission on NIC 2 (see Fig. 2), and vice versa. The Ryu script was also configured to periodically issue requests to Lagopus to obtain OpenFlow statistics.

Next, the applications were started. Based on the experiment, one or both `iperf3` and `ping` applications were executed. The log files were collected from these applications. Python and R scripts were used to extract application-level measurements from the `iperf3` and `ping` logs. For Lagopus switching performance, Ryu logs were parsed to extract forwarding and drop rates.

C. Application performance under low utilization

To study packet delays when the link is under-utilized, we executed three runs on the InstaGENI setup: (i) `ping` flow only, (ii) `iperf3` flow only, and (iii) `ping` and `iperf3` flows together. In run 1, the duration of the `ping` flow was 100 sec, which was also the duration of the `iperf3` flow

Table II
IMPACT OF PACKET BATCHING; NA: NOT APPLICABLE

Loop count (pkts)	Application flows	Delay (ms)		Throughput (Mbps)	
		mean	SD	mean	SD
10000	`ping` only	4.468	1.159	NA	NA
100	`ping` only	0.507	0.447	NA	NA
10000	`iperf3` only	NA	NA	939.64	5.51
100	`iperf3` only	NA	NA	939.77	6.97
10000	both	See Fig. 3		939.51	5.70
100	both	See Fig. 3		939.09	5.57

in run 2. In run 3, the `ping` flow was started first, and then the `iperf3` flow was started 10 sec later. The duration of the `iperf3` flow was 100 sec. The ping flow ran for an additional 10 sec after the `iperf3` flow ended.

In all runs, the number of worker threads was set to 4, but only 1 worker thread was used. This is because in all cases, packets were sent for all flows from host H1 to host H2, which caused all packets to be sent to the same worker thread given the load balancing approach described in Section III-C.

Table II shows the results. Both `ping` and `iperf3` applications were configured to report delay and throughput, respectively, every sec. Therefore, the mean and standard deviation (SD) values shown in Table II are across the multiple per-sec reports obtained in the `ping` and `iperf3` logs. There were no losses in the `ping` flows, and no packet retransmissions in the `iperf3` flows.

Since packets were sent at intervals of 1 sec in the `ping`-only run 1, the time to fill the 144-packet buffer (see Section III-D) exceeded the time to execute the I/O receive and I/O transmit loops even at the 10000 loop count setting. Therefore the packet delays for the `ping` flow were higher with the I/O thread loop count set to 10000 when compared to the 100 setting as seen in rows 1 and 2 of Table II.

Rows 3 and 4 show that the loop count did not impact `iperf3` flow throughput. This is because Lagopus is able to forward packets at 1 Gbps rates in this experimental setup. The last two rows show that `iperf3` throughput is unaffected by the presence of the `ping` flow. However, the reverse is not true.

Fig. 3 shows that in the presence of a high-rate flow (i.e., `iperf3` flow), the delay suffered by ping packets increases with time. The increasing delays experienced by ping packets indicates that the overall processing rate is slower than the packet arrival rate. The difference in these rates is reflected in the slope of the linear increase observed in Fig. 3. When the `iperf3` flow ended, packet delays in the `ping` flow dropped to their original values. As seen with the run 1, when the only flow present was the `ping` flow, packet delays were higher at the larger loop count setting. The higher variability in packet delay when the loop count is 10000 can be attributed to the variability in the time instant at which packets arrive relative to the start of each loop. In other words, if a packet arrived just before the end of a 10000-count loop, it will suffer lower delay than if a packet arrived soon after the beginning of a loop.

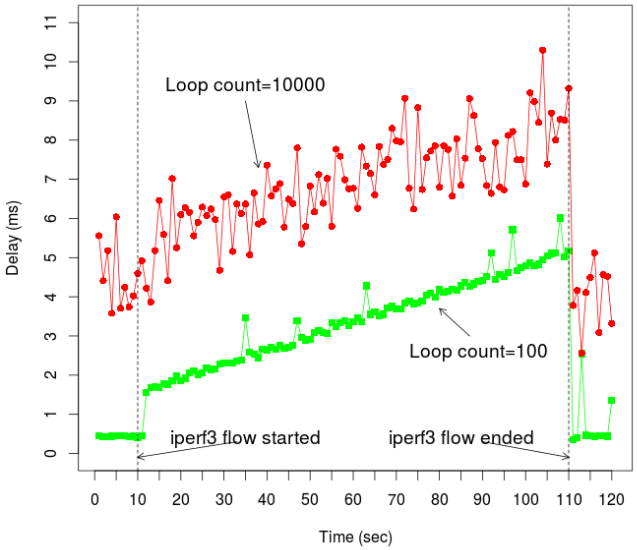

Figure 3. Impact of high-rate flow on a delay-sensitive flow

This experiment demonstrates the need for packet classification at the receive thread to direct delay-sensitive flows to a different worker thread from high-throughput flows. Further, a small loop count is required to handle light traffic loads.

D. Impact of parallelization

In the experiments described in Section IV-C, packets from both the `ping` and `iperf3` flows were sent by the receive thread to the same worker thread. We designed a new experiment to test the impact of parallelization by creating flows that would be directed to multiple worker threads. This experiment was executed on the InstaGENI setup.

Given the packet classification approach used in Lagopus 0.2.2 as described in Section III-C, two `iperf3` flows were sent in opposite directions, H1 $\rightarrow$ H2 and H2 $\rightarrow$ H1. Since the receive thread uses source and destination MAC addresses to distribute flows to worker threads, packets from these two flows were sent to two separate worker threads.

The I/O thread loop count was set to 100 packets. The number of OpenFlow table entries was set to 1000 to increase the worker-thread processing load. This change was required because with the previous setting of three OpenFlow table entries, as described in Section IV-B, the benefit of multiple worker threads could not be demonstrated as a single worker thread could easily handle two 1-Gbps `iperf3` flows. With 1000 OpenFlow table entries that required matching on the Destination MAC address, worker-thread processing became the bottleneck, allowing for a demonstration of the value of multiple worker threads.

The Lagopus version used in this work, 0.2.2, used Patricia tries for flow table lookup as noted in Section III-E. This trie

Table III
IMPACT OF THE NUMBER OF WORKER THREADS

No. of worker threads	No. of OF table entries	No. of flows	Flow (iperf3)	Throughput (Mbps)	
				mean	SD
1	3	1	H1 → H2	939.5	6.9
1	3	2	H1 → H2	470	6.1
	3		H2 → H1	470	6.2
2	3	2	H1 → H2	935.58	5.74
	3		H2 → H1	935.54	5.4
1	1000	1	H1 → H2	427.5	12.8
1	1000	2	H1 → H2	184	36.1
	1000		H2 → H1	245.5	34.6
2	1000	2	H1 → H2	398.5	42.6
	1000		H2 → H1	390.4	40.13

solution is not suitable for 64-bit MAC address based lookups. Other lookup methods such as hash tables are currently being added to support MAC-address based L2 packet forwarding in new releases of Lagopus.

The two `iperf3` flows were run simultaneously for 100 sec, each, and 100 per-sec values of `iperf3` throughput were obtained from the log files. The mean and SD values are shown for different settings in Table III.

The results in Table III show that a single worker thread was capable of handling two `iperf3` flows at close to 1 Gbps when the number of OpenFlow table entries was only 3, but with 1000 destination-MAC address based entries, a single worker thread could handle packets from a single flow at approximately 427 Mbps. When the number of OpenFlow table entries was 3, when the two flows were handled by the same worker thread, each flow received only 470 Mbps, but when each flow was handled by a separate worker thread, throughput was 935 Mbps. With 1000 OpenFlow entries, a similar effect was observed, with an average of 214 Mbps when both flows were handled by the same worker thread, and an average of 394 Mbps when the flows were handled by separate worker threads.

This experiment demonstrated that the use of multiple worker threads can improve performance significantly. Therefore, the receive thread should support more sophisticated load balancing approaches in addition to the current solution of using destination and source MAC addresses and the next four bytes for worker thread selection.

E. High-throughput performance

This experiment used the Keio setup as described in Section IV-A with Host 1 running Lagopus and Host 2 running `pktgen` to create MPLS packets. As we did not have a hardware tester, we could use only this software packet generator. It was capable of generating packets at 10 Gbps when the packet size was 1500B, but the rate was lower for smaller packets. Since a prior paper [3] documented the impact of packet size, here we focused on packet drop rate.

The `pktgen` program was provided 1500B as a runtime argument. This program uses this argument to set the maximum length of the Ethernet frame sent on the wire. Since the MPLS header is 4 bytes, and Ethernet header is 14 bytes, the `pktgen` program set the IP payload to 1482B (which was determined from a packet-capture (pcap) file). The `pktgen` program was run for 1 min, and as the NIC rate was 10 Gbps, 50M packets were sent from Host 2 to Host 1. OpenFlow (OF) table lookup used MPLS label for matching, and the action was simple forwarding.

Table IV
IMPACT OF OPENFLOW (OF) TABLE SIZE

No. of OF table entries	No. of worker threads	Receive rate (Gbps)	Transmit (Forwarding) rate (Gbps)	Packet drop rate (%)
100K	1	9.810	9.808	0.002
100K	4	9.745	9.742	0.003
400K	1	9.813	9.809	0.03
400K	4	9.814	9.755	0.6
1M	1	9.823	9.797	0.27
1M	4	9.815	9.513	3.0

The number of worker threads used was set to 1 or 4, and the I/O thread loop count was 10000 packets. The remaining parameters were set to the values shown in Table I. The receive thread, worker threads and transmit thread were placed in different cores in the same socket. The OF table size was varied to study its impact on packet drop rate. With four worker threads, packets were uniformly distributed to all the worker threads. Lagopus has a configurable parameter for sending all packets of a flow to the same worker thread as described before, but this parameter was disabled for this experiment so that packets were distributed to all four worker threads.

Table IV shows results obtained from the statistics collected every 10 sec by Ryu (byte counts and packet counts). Since packets were generated for 1 min, six observations were obtained for incoming and outgoing byte and packet counts. The values shown for receive rate and transmit rate (which corresponds to forwarding rate) represent mean values computed from the six observations for each setting of the OF table size. The packet drop rate column shows a surprising result.

The packet drop rate was higher with four worker threads than with one worker thread. Our analysis shows that this occurs because the transmit thread incurs an overhead when reading packets from four ring buffers, i.e., one buffer per worker thread. In other words, the average per-packet processing time is higher with four worker threads than with a single worker thread. The (single) transmit thread is more likely to find a higher number of packets in the output ring buffer when a single worker thread is used, than in any single ring buffer when four worker threads are used. The overhead of reading packets from ring buffers is amortized over a larger number of packets with a single worker thread than with four worker threads. This explains why the average per-packet processing time is higher with four worker threads than with a single worker thread. Since packet losses can lower throughput of TCP flows on high bandwidth-delay product

paths significantly, it is important to limit packet arrival rate to a Lagopus switch to ensure zero packet drop rate.

This experiment presented the impact of parallelization on switch performance, which includes both packet forwarding rate and packet drop rate, at high loads in a configuration with high link rates.

V. Summary and Conclusions

Our first recommendation is that packet classification should be done in the receive thread to direct packets from delay-sensitive flows to a different worker thread from that used for high-throughput flows. Our second finding is that a small loop count should be used in packet batching especially when link utilization is low. Our third finding is that the receive thread should support additional load balancing options besides the current solution of using destination and source MAC addresses and the next four bytes for worker thread selection. Our fourth finding is that at high packet arrival rates (10 Gbps), if the size of the OpenFlow table is large, e.g., 1M entries, packet drop rate was 0.27% with one worker thread. The packet drop rate increased to 3% with four worker threads because the average per-packet processing delay was higher than with one worker thread. Therefore, a software switch should be calibrated for the highest packet arrival rate that it can handle with zero packet drops.

VI. Acknowledgment

This work was supported by NSF CNS-1116081, OCI-1127340, ACI-1340910, CNS-1405171, and CNS-1531065, and U.S. DOE grant DE-SC0011358. The Keio University work was supported by the "ACTION Project" funded by the National Institute of Information and Communications Technology (NICT) Japan.

References

[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Denf *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[2] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *Communications Surveys Tutorials, IEEE*, vol. 18, no. 1, pp. 236–262, 2016.

[3] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui. Scalable, High-performance, Elastic Software OpenFlow Switch in Userspace for Wide-area Network. [Online]. Available: https://www.usenix.org/conference/ons2014/poster-session

[4] Open vSwitch. [Online]. Available: http://openvswitch.org

[5] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15, 2015.

[6] K. Fall, G. Iannaccone, M. Manesh, S. Ratnasamy, K. Argyraki, M. Dobrescu, and N. Egi, "Routebricks: Enabling general purpose network infrastructure," *SIGOPS Oper. Syst. Rev.*, vol. 45, no. 1, pp. 112–125, Feb. 2011.

[7] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.

[8] Agilent N2X router tester. [Online]. Available: http://www.ixiacom.com/products/ixn2x

[9] G. Pongracz, L. Molnar, and Z. L. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*. IEEE, 2013, pp. 62–67.

[10] Intel® Data Plane Development Kit (DPDK). [Online]. Available: http://dpdk.org/

[11] D. Scholz, "A look at Intel's Dataplane Development Kit," *Network*, vol. 115, 2014.

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

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[14] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '15, 2015.

[15] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 120–125.

[16] Open vSwitch accelerated by Intel® DPDK. [Online]. Available: https://github.com/01org/dpdk-ovs

[17] Y. Nakajima. (2014, Nov.) Lagopus: High-performance software OpenFlow switch for wide-area network. [Online]. Available: http://www.itsa.org.tw/site2014/wp-content/uploads/20141114-lagopus-taiwan-handson-2014-november-printout.pdf

[18] Intel® Data Direct I/O Technology. [Online]. Available: http://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html

[19] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann Series in Networking, 2005.

[20] J. D. Valois, "Implementing lock-free queues," in *In Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems, Las Vegas, NV*, 1994, pp. 64–69.

[21] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014, special issue on Future Internet Testbeds Part I. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613004507

[22] iperf3. [Online]. Available: http://software.es.net/iperf/

[23] Ryu: A software defined networking framework. [Online]. Available: https://osrg.github.io/ryu/

[24] Linux Foundation: pktgen. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen

[25] DPDK IXGBE Driver. [Online]. Available: http://dpdk.org/doc/guides/nics/ixgbe.html