Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches

Frank Dürr, Thomas Kohler
Institute of Parallel and Distributed Systems (IPVS)
Universität Stuttgart
Stuttgart, Germany
{frank.duerr,thomas.kohler}@ipvs.uni-stuttgart.de

TR 2014/04

Abstract

In this report, we compare the forwarding latency of an OpenFlow hardware switch and software switch, namely, the hardware switch NEC PF5240 and the software switch Open vSwitch. We consider different parameters influencing the forwarding latency such as the definition of match tuples (combination of header fields to be matched), different packet sizes, or varying sizes of flow tables. Our evaluations show that hardware switches reduce the forwarding latency by up to 97% and also reduce the variance of latency significantly.

1 Introduction

Using hardware switches in a software-defined network (SDN) is very attractive since it promises high forwarding performance, namely, low latency and line-rate throughput, without sacrificing the benefits of SDN, in particular, the flexible control of the network through an SDN network controller, which is programming the forwarding tables of switches. Therefore, hardware acceleration is one prerequisite to enable applications relying on high throughput and low latency such as high-frequency trading and systems supporting such applications like the low-latency message-oriented middleware developed at University of Stuttgart [2].

The goal of this report is to evaluate and compare the performance in terms of forwarding latency of state-of-the art hardware and software switches. For our experiments, we have chosen two typical representatives of both classes, namely, the hardware switch NEC PF5240 and the software switch Open vSwitch. Due to the support of dedicated hardware like ternary content-addressable memory (TCAM) for flow table lookups, we expect a significant performance improvement of the hardware switch over the software switch.

In our evaluations, we consider different influencing parameters like the definition of match tuples of flow table entries, different packet sizes, or different sizes of flow tables.

The rest of this report is structured as follows. First, we introduce the methodology used for our experiments in Section 2. Then, we introduce the systems under test in Section 3. In Section 4, we present the results of our evaluations, before we conclude this report in Section 5.

2 Methodology

Before we present our measurements, we first introduce basic definitions, state our assumptions, and describe the methodology and technologies used for measuring the forwarding latency.

2.1 Calculation of Processing Delay

First of all, it is important to define the different kinds of delays that occur in the network between sending and receiving a packet:

- Propagation delay (Δt_{prop}): The propagation delay denotes the delay of the signal in the physical medium (twisted pair copper wires in our experiments). We assume a signal speed of two third of the speed of light, i.e., $\frac{2}{3} \times 3 \times 10^8 \frac{\text{m}}{\text{s}}$.
- Transmission[1] delay (Δt_{trans}): The transmission delay is the time to "put the data on the wire". This delay depends on the frame size s and the data rate r of the link (1 Gbps in our experiments): $\Delta t_{\text{trans}} = \frac{s}{r}$
- Queuing delay (Δt_{queue}): The queuing delay denotes the time a packet spends in a queue of the switch, e.g., if there are multiple packets waiting to be transmitted over the same outgoing port.
- Processing delay (Δt_{proc}): The processing delay defines the time it takes the switch to make a forwarding decision to decide over which outgoing port to forward a packet.

We are mainly interested in the processing delay Δt_{proc} since we expect the biggest influence of hardware acceleration there. However, we also have to consider the other delays to calculate the processing delay.

In our experiments, we measure the round-trip time Δt_{rtt} of packets that are looped back by the switch to the sending host. Figure 1 depicts the delays involved in a single measurement as performed in our setup.

We send packets at time intervals of 500 ms, which even for the maximum frame size (MTU 1500 byte) results in a data rate much below line rate (1 Gbps in

[1] Also called "serialization delay".

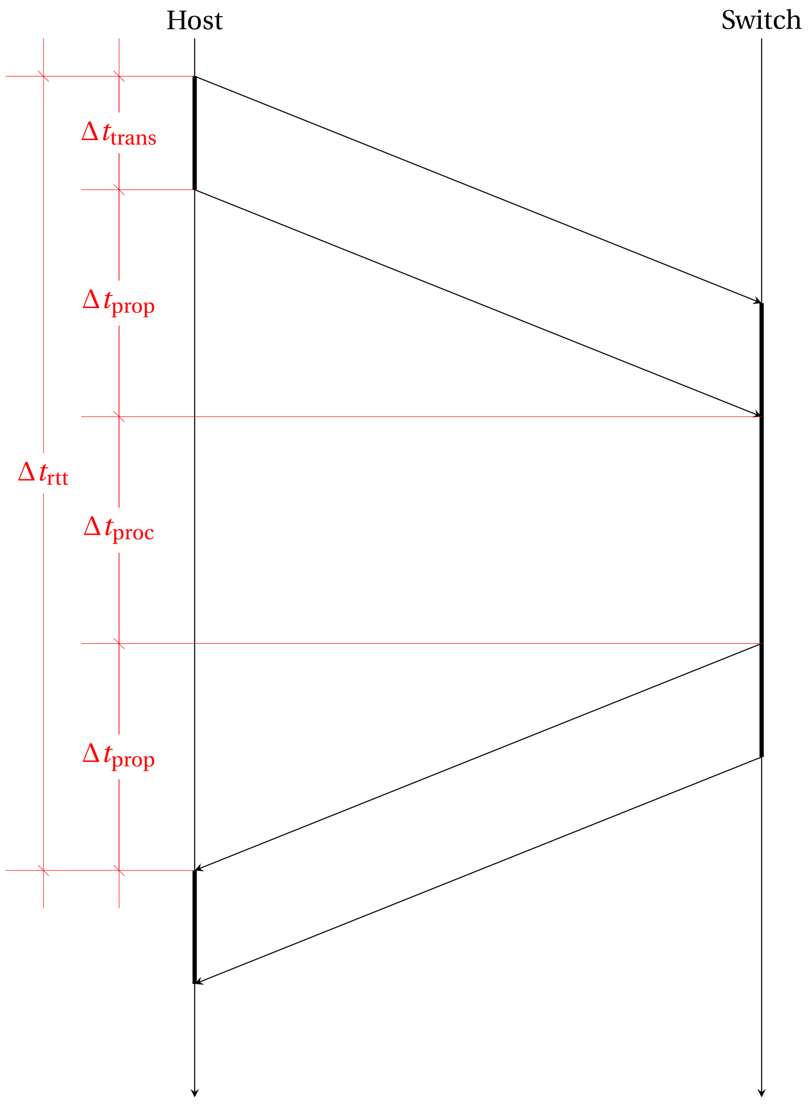

Figure 1: Delays of one round-trip time measurement. Δt_{queue} is assumed to be 0 and, therefore, not depicted.

our system). Therefore, packets should not queue up in the switch, and, consequently, we assume a queuing delay of $\Delta t_{\text{queue}} = 0\,\text{s}$.

Depending on the setup (cf. Section 3), we use 1 m and 3 m copper (twisted-pair) cables to connect the NIC to the switch. Therefore, the propagation delay is calculated as $\Delta t_{\text{prop}} = \frac{1\,\text{m}}{\frac{2}{3}\times 3\times 10^8\,\frac{\text{m}}{\text{s}}} = 5\,\text{ns}$ and $\Delta t_{\text{prop}} = \frac{3\,\text{m}}{\frac{2}{3}\times 3\times 10^8\,\frac{\text{m}}{\text{s}}} = 15\,\text{ns}$. Since we perform round trip measurements, the propagation delay has to be considered twice in one round-trip measure (cf. Figure 1).

We assume that packets are timestamped at the latest point in time when they leave the NIC and the earliest point in time when they enter the NIC (cf. Figure 1). To this end, we use a NIC with hardware timestamping support (see Section 2.2). Actually, timestamps are generated by the NIC after the preamble and start of frame delimiter of a MAC frame [1]. However, since the NIC generates timestamps at exactly the same positions within a frame, this will not change our calculations, but only shifts the time interval Δt_{rtt} by a constant offset along the time axis.

We assume a "store-and-forward" behavior of the switches rather than "cut-through" forwarding. Therefore, before making a forwarding decision, the switch first receives the complete frame. As can be seen from Figure 1, this leads to the following equation defining the relations between the measured round trip time and the various delays:

$$\Delta t_{\text{rtt}} = \Delta t_{\text{proc}} + \Delta t_{\text{queue}} + 2\Delta t_{\text{prop}} + \Delta t_{\text{trans}} \tag{1}$$

According to our assumptions, $\Delta t_{\text{queue}} = 0\,\text{s}$. Under this assumption, the processing delay can be calculated as

$$\Delta t_{\text{proc}} = \Delta t_{\text{rtt}} - 2\Delta t_{\text{prop}} - \Delta t_{\text{trans}} \tag{2}$$

2.2 Measuring the Round Trip Time with Hardware Timestamping

Measuring the round trip time Δt_{rtt} accurately and precisely is challenging for time intervals in the order of microseconds. In order to be able to measure such short delays, we use hardware timestamping with support of the network interface controller (NIC) of the host rather than software timestamping performed by the driver in kernel space or even the application in user space. The main advantage of hardware timestamps compared to software timestamps is that packets are timestamped by the NIC hardware at precisely defined points in time (after the preamble and start of frame delimiter) when they leave or enter the NIC. Thus, delays introduced by packet processing in the kernel space or user space of the host are excluded from the measurements.

Note that timestamping a packet does not mean that a timestamp is included in the packet sent over the wire. The sending process receives the timestamp when the packet was sent via the sender socket (in Linux via the socket's error queue). Similarly, the receiving process receives the timestamp of the received packet via the receiver socket.

Moreover, we use the same clock of the same NIC to create timestamps to avoid difficulties of clock synchronization. Since packets to destinations on the same host (local IP address; 10.0.0.1 in our experiments) are directly delivered to the receiver process via the kernel rather than sending it out over the NIC[2], we send packets to other IP addresses (different from the local address 10.0.0.1). Such packets with non-local addresses are forwarded to the switch, which loops them back to the host using the OpenFlow LOOPBACK action. The NIC of the host is configured in promiscous mode such that looped-back packets will be accepted and timestamped although they are targeted at another address. We set static ARP table entries on the host for each destination IP address. Thus, no ARP requests are performed and only the UDP datagrams (probe packets) of the measurements are sent over the wire.

3 Systems under Test

We use the following hardware and software in our tests:

- Hardware switch: NEC PF5240 (48 port Gigabit switch implementing OpenFlow 1.0 and 1.3)
- Software switch: Open vSwitch 1.10.0
- Host 1 running sender/receiver process and Open vSwitch
  - CPU: Intel i7-2600, 3.40 GHz
  - RAM: 8 GB
  - NIC: Intel I350 (four port Gigabit Ethernet) with hardware timestamping support
  - Operating system: Fedora 3.10.6-100.fc18.x86_64
- Host 2 running sender/receiver process; connected to hardware switch
  - CPU: Intel Xeon E3-1245 V2, 3.40 GHz
  - RAM: 16 GB
  - NIC: Intel I350 (four port Gigabit Ethernet) with hardware timestamping support
  - Operating system: Fedora 3.13.9-200.fc20.x86_64
- OpenFlow controller: OpenDaylight (Hydrogen release)

[2] Using forwarding rules, this behaviour could be changed under Linux.

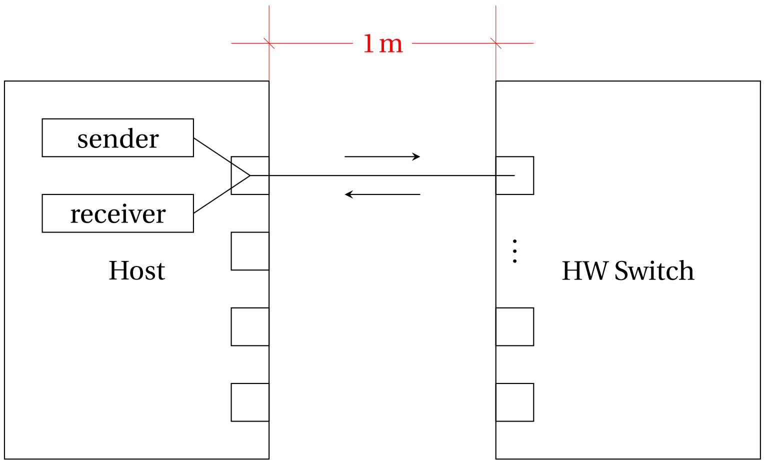

Figure 2: Setup hardware switch

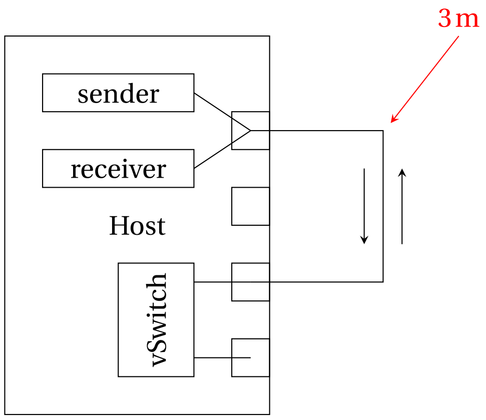

Figure 3: Setup software switch

Figure 2 depicts the setup for the hardware switch. The switch is connected to the host via a 1 m twisted pair cable using one port of the 4-port NIC. The OpenFlow controller runs on a dedicated host using a dedicated control network (not shown in the figure).

Sender and receiver are two processes implemented in C++ collecting timestamps for each UDP datagram sent and received. A Python script controls the experiment.

Figure 3 depicts the setup for the software switch (Open vSwitch). The software switch is running on the same host as the sender and receiver and is assigned one port of the 4-port NIC. This port is connected to another port of the 4-port NIC used by the sender and receiver processes with a 3 m twisted pair cable.

4 Results

In this section, we present the results of our performance evaluation. In each experiment, we measure the round trip time and calculate the processing time while varying different parameters. The complete data sets are available on request from the authors.

4.1 Varying Packet Size

In the first experiment, we vary the packet size. The packet size directly influences the transmission delay while the propagation delay should stay constant. Since the time for making a forwarding decision is also independent of the packet size, we also expect the processing delay to be nearly constant. Since we only perform black box tests, we cannot say if there are size-dependent operations like copy operations of frames between buffers. Therefore, this experiment will uncover whether the processing delay is also size-dependent.

We measured the round trip times for eight different packet sizes leading to the following eight frame sizes ranging from the minimum allowed Ethernet framesize up to the maximum frame size for an MTU of 1500 byte: 72 byte, 254 byte, 454 byte, 654 byte, 854 byte, 1054 byte, 1254 byte, and 1526 byte. These sizes includes the layer 2-4 headers and payload of the UDP datagram (8 byte UDP header, 20 byte IPv4 header, 26 byte Ethernet header, UDP payload)

We program the switch with 200 flow table entries. Each entry includes a match on a different destination MAC address. We send UDP datagrams to random IP addresses each one associated with one of the 200 MAC addresses of the installed flows. We send 10000 datagrams and calculate the average round trip time and standard deviation as well as the minimum round trip time and average processing delay.

Table 1 and Table 2 show the results for the hardware switch and software switch, respectively. Figure 4 depicts the processing delay of the hardware and software switch.

For the hardware switch, we measure almost the same average processing delays for the maximum framesize of 1526 bytes and a framesize of 254 bytes. Therefore, we conclude that the processing delay is independent of the framesize, and the measured differences for different frame sizes (max. 1.0 μs) is be due to measurement imprecisions.

For the software switch, the processing delay increases from 123055.9 ns to 125329.3 ns. However, the difference in processing the smallest and largest frames is only 2273.4 ns. Thus, also for the software switch, the framesize has little impact on the processing delay considering the overall processing delay.

On average, the processing delay of the hardware switch is only 3.1% of the processing delay of the software switch for 72 byte frames showing that the hardware switch is significantly faster than the software switch.

Moreover, the standard deviation of the round trip times for the hardware

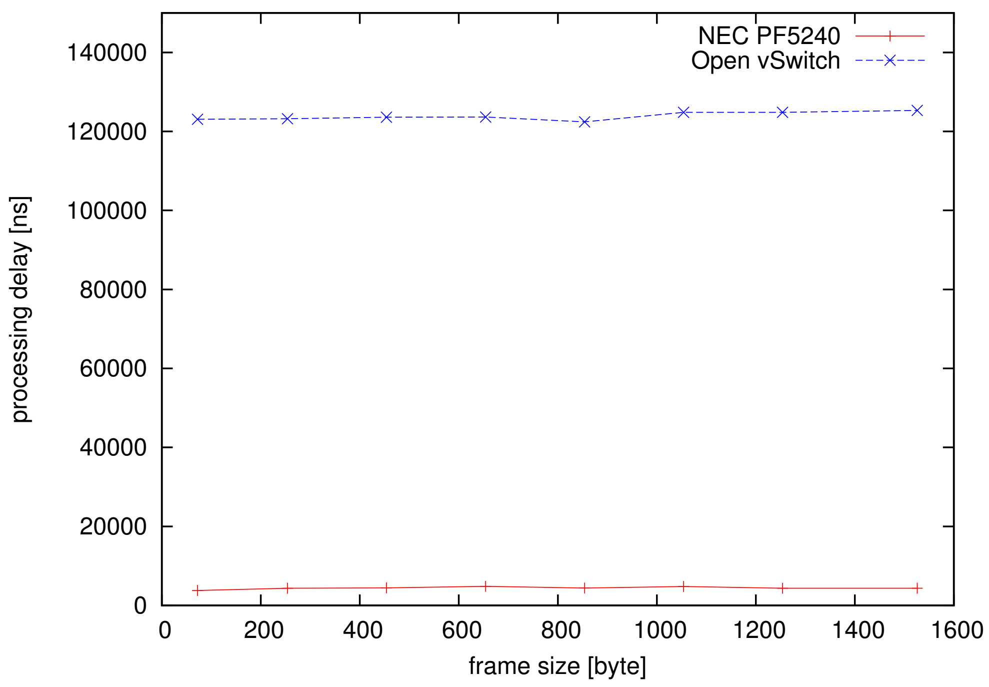

Figure 4: Processing delay for varying frame sizes.

switch is very small (less than 161.8 ns) compared to the software switch, which has a significantly larger standard deviation up to 45327.23 ns.

4.2 Varying Match Tuples

In the next experiment, we are interested whether different matching criteria in flow table entries lead to different processing delays. We consider three different match tuples:

1. Layer 2 matching: ingress port, data link layer destination address
2. Layer 3 exact match: ingress port, ether type, network layer destination address (IPv4; exact match)
3. Layer 3 prefix match: ingress port, ether type, network layer destination address (IPv4; prefix/subnet match)
4. Layer 2-4 matching (9 tuple): ingress port, ether type, data link layer source and destination addresses, network layer source and destination addresses, source and destination port, layer 4 protocol id
5. Mix: A random mixture of the cases Layer 2, Layer 3 (exact match), Layer 2-4 described above.

We send 10000 minimum size frames of 72 byte. The flow table contains 200 entries.

Table 3 and Table 4 show the average round trip times and processing delays for the hardware and software switch, respectively. For both hardware and software switch, the tuple combination has almost no influence on the processing delay. Only the prefix match takes slightly longer (444.0 ns longer) than the other match tuples on the hardware switch. For the software switch, the maximum difference is 1432.8 ns, which again is very small compared to the absolute processing delay of the software switch.

4.3 Varying Flow Table Size

Finally, we analyze the latency of forwarding for a varying number of flow table entries.

We send frames of the minimum size of 72 byte and vary the flow table size from 200 to 100000 entries for the software switch. The hardware switch is restricted in the maximum size of the forwarding table. Therefore, the maximum flow table size tested for the hardware switch in our experiments is 45000 entries.

Flow table entries use the network destination (IPv4) address (exact match), ether type, and ingress port as match criteria. For each flow table size, we send 10000 UDP datagrams and calculate the average round trip time, minimum round trip time, standard deviation, and processing delay as shown in Table 5 and Table 6 for the hardware and software switch, respectively. Figure 5 and Figure 6 depict the processing delays for different flow table sizes for both switches.

As can be seen from Figure 5, the processing delay of the hardware switch is independent of the size of the forwarding table. Moreover, in contrast to the software-switch, the standard deviation of the processing delay stays constant over the whole range of flow table sizes.

For the software switch, the processing delay increases very slowly as can be seen from Figure 6: the minimum and maximum average processing delay only differ by about 10%. However, the standard deviation increases significantly from 42644.22 ns (200 entries) to 659207.8 ns (100000 entries).

Therefore, we conclude that both types of switches scale well with the number of forwarding table entries. Again, hardware support leads to much smaller forwarding latencies. Additionally, the variance of processing latency is reduced significantly for large forwarding tables using hardware switches.

5 Conclusion

Low latency is an essential requirement for many networked applications. In our evaluations we compared the processing delay of the hardware switch NEC PF5240 and the software switch Open vSwitch. Our evaluations showed that the hardware switch leads to significantly lower processing delay. In our exper-

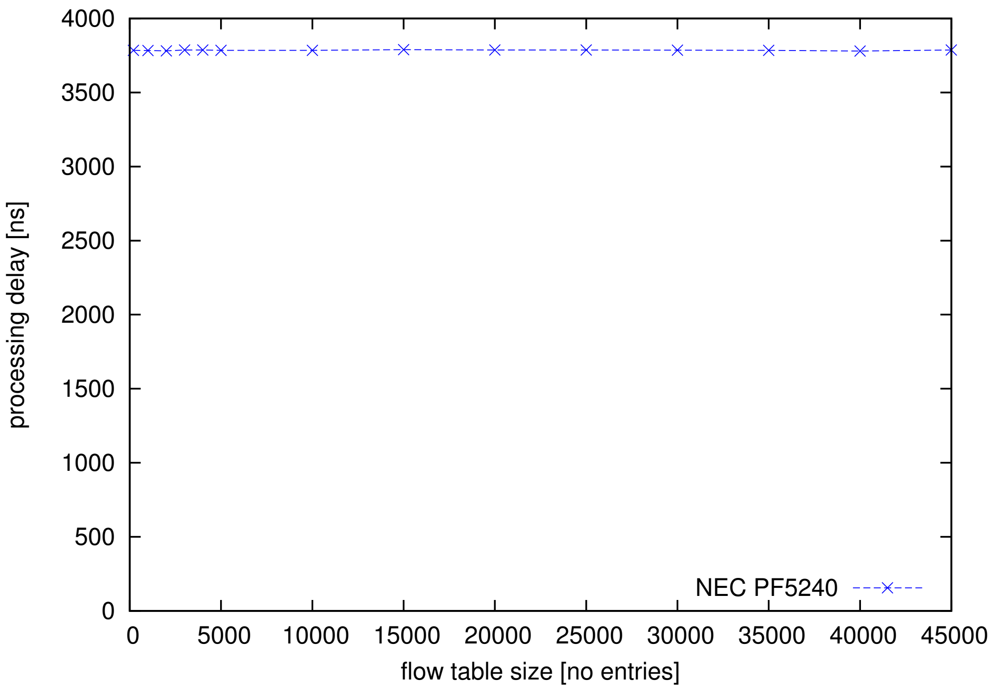

Figure 5: Processing delay for varying flow table sizes: hardware switch (NEC PF5240)

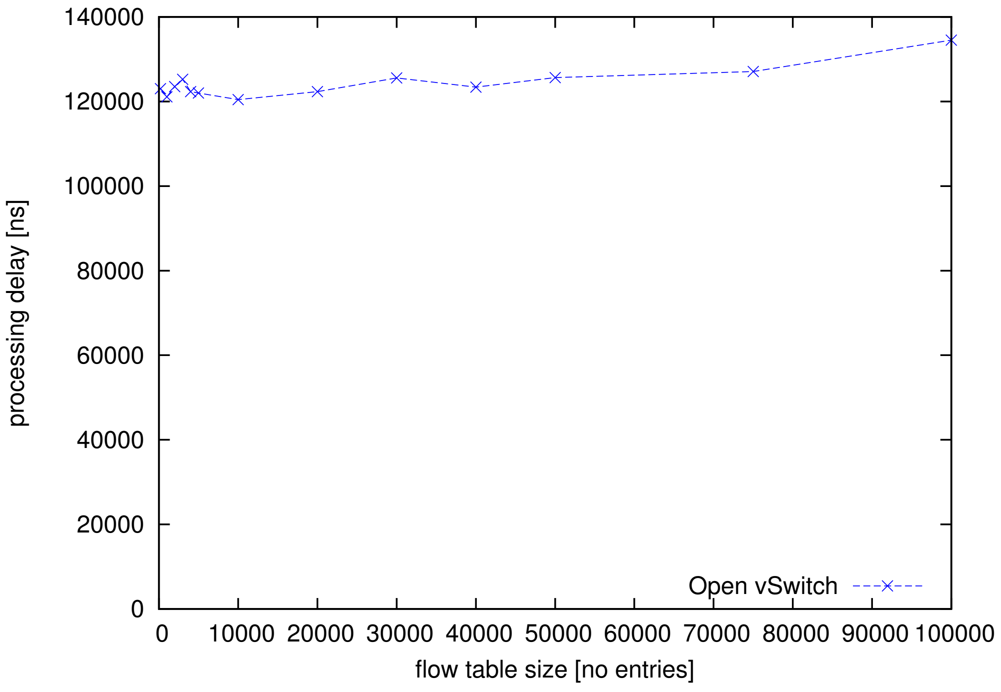

Figure 6: Processing delay for varying flow table sizes: software switch (Open vSwitch)

iments, the hardware switch reached processing delays of about 3.8 μs on average, whereas the average processing delay of the software switch was higher than 120 μs. Thus, hardware support reduced processing time by about 97%.

Moreover, the hardware switch showed very constant performance with low delay variance, whereas the delay variance for the software switch is much higher, in particular, for larger flow tables.

We also found out that the matching criteria (tuples) has almost no influence on the forwarding performance (for the tested tuples, including the most relevant matches on MAC addresses and IP addresses).

Finally, we showed that both hardware and software switches scale well with the number of forwarding table entries. Obviously, the maximum number of forwarding table entries is limited for the hardware switch (about 50000 IPv4 entries in our case), whereas a software switch can support hundreds of thousands of entries. However, in the range supported by the hardware switch, the forwarding latency and variance of latency is much lower using hardware switches than for software switches.

Several other interesting properties of OpenFlow switches remain open for further evaluations. For instance, it might be interesting to evaluate the throughput and latency of forwarding table updates of software and hardware switches.

References

[1] Intel Corporation. *Intel Ethernet Controller I350 Datasheet*, Mar. 2013.

[2] B. Koldehofe, F. Dürr, M. A. Tariq, and K. Rothermel. The power of software-defined networking: Line-rate content-based routing using openflow. In *Proceedings of the 7th Middleware for Next Generation Internet Computing Workshop of the 13th International Middleware Conference 2012 (MW4NG 2012)*, Montreal, Canada, Dec. 2012.

frame size [byte]	calculated Δ_{trans} [ns]	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
72	576	4371.641	160.8093	4064	3785
254	2032	6380.42	161.8341	6080	4338
454	3632	8092.177	159.9805	7776	4450.2
654	5232	10057.55	159.3937	9744	4815.6
854	6832	11213.52	158.9263	10912	4371.5
1054	8432	13212.6	161.7336	12904	4770.6
1254	10032	14382.22	161.2278	14088	4340.2
1526	12208	16556.64	160.4316	16264	4338.6

Table 1: Delays for varying frame sizes: hardware switch (NEC PF5240)

frame size [byte]	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
72	123661.9	43725.75	31288	123055.9
254	125263.7	43737.64	33800	123201.7
454	127272.3	43077.26	35992	123610.3
654	128892.4	42620.84	38136	123630.4
854	129274.8	42568.54	39048	122412.8
1054	133288.5	44118.43	41768	124826.5
1254	134896.3	44417.49	41896	124834.3
1526	137567.3	45327.23	44712	125329.3

Table 2: Delays for varying frame sizes: software switch (Open vSwitch)

match tuple	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
Layer 2	4369.705	160.3563	4064	3783.7
Layer 3 (exact)	4375.251	160.1	4064	3789.3
Layer 3 (prefix)	4813.69	160.5379	4512	4227.7
Layer 2-4 (9 tuple)	4369.727	159.9956	4064	3783.7
Mix	4372.987	159.5453	4064	3787.0

Table 3: Delays for different match tuples: hardware switch (NEC PF5240)

match tuple	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
Layer 2	123661.9	43725.75	31288	123055.9
Layer 3 (exact)	123399.8	42869.42	27480	122793
Layer 3 (prefix)	122782.4	43486.34	31512	122176.4
Layer 2-4 (9 tuple)	122254.5	43141.94	32072	121648.5
Mix	123687.2	43968	31432	123081.2

Table 4: Delays for different match tuples: software switch (Open vSwitch)

flow table size [no. entries]	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
200	4370.806	161.1272	4064	3784.8
1000	4369.468	159.5622	4064	3783.5
2000	4367.302	159.8656	4064	3781.3
3000	4373.164	162.5401	4064	3787.2
4000	4372.374	160.5801	4064	3786.4
5000	4370.516	161.4779	4064	3784.5
10000	4370.312	160.6101	4064	3784.3
15000	4374.647	161.2364	4064	3788.6
20000	4372.254	161.8506	4064	3786.3
25000	4372.905	160.176	4064	3786.9
30000	4371.989	160.6105	4064	3786.0
35000	4370.72	161.299	4064	3784.7
40000	4366.06	161.5515	4064	3780.1
45000	4372.78	160.5618	4064	3786.8

Table 5: Delays for varying flow table sizes: hardware switch (NEC PF5240)

flow table size [no. entries]	mean Δt_{rtt} [ns]	standard deviation Δt_{rtt} [ns]	minimum Δt_{rtt} [ns]	mean Δt_{proc} [ns]
200	123636.3	42644.22	31176	123030.3
1000	121726.6	43853.06	31320	121120
2000	124125.6	44085.8	31560	123519.6
3000	125838.5	44150.11	31240	125232.5
4000	122921.6	43455.8	30376	122315.6
5000	122589.8	43555.71	31576	121983.8
10000	121037	42907.3	31368	120431
20000	122956.4	43616.56	31448	122350.4
30000	126183.7	96816.4	31176	125577.7
40000	124006.5	116279.7	31400	123400.5
50000	126241.6	193462.6	31464	125635.6
75000	127701.7	165940.3	31496	127095.7
100000	135123.5	659207.8	31576	134517.5

Table 6: Delays for varying flow table sizes: software switch (Open vSwitch)