Using CPU as a Traffic Co-processing Unit in Commodity Switches

Guohan Lu
Microsoft Research Asia
Beijing, China
lguohan@microsoft.com

Rui Miao[*]
Tsinghua University
Beijing, China
rm870725@gmail.com

Yongqiang Xiong
Microsoft Research Asia
Beijing, China
yqx@microsoft.com

Chuanxiong Guo
Microsoft Research Asia
Beijing, China
chguo@microsoft.com

ABSTRACT

Commodity switches are becoming increasingly important as they are the basic building blocks for the enterprise and data center networks. With the availability of all-in-one switching ASICs, these switches almost universally adopt single switching ASIC design. However, such design also brings two major limitations, *i.e*, limited forwarding table for flow-based forwarding scheme such as Openflow and shallow buffer for bursty traffic pattern. In this paper, we propose to use CPU in the switches to handle not only control plane but also data plane traffic. We show that this design can provide large forwarding table for flow-based forwarding scheme and deep packet buffer for bursty traffic. We build such a prototype switch on ServerSwitch platform. In our evaluation, we show that our prototype can achieve over 90% traffic offloading ratio, absorb large traffic bursts without a single packet drop, and can be easily programmed to detect and defend low-rate burst attacks.

Categories and Subject Descriptors

C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design—*Packet-switching networks*

General Terms

Design

Keywords

Traffic Co-processing Unit, Commodity switch, Large forwarding table, Deep buffer

[*]This work was performed when Rui Miao was a visiting student at Microsoft Research Asia.

1. INTRODUCTION

Commodity switches are the basic building blocks of enterprise and data center networks (DCN). Usually with 48 or 64 Ethernet ports, thousands or even more of these boxes are deployed as top of rack (ToR) switches in large scale data centers. Moreover, these boxes are becoming increasingly important as recent researches [1, 2] suggest that the entire DCN can be built using these boxes for better scalability and significant cost savings.

There are two design trends for these commodity switches. First, most of ToR switches now adopt single switching ASIC approach [3], *i.e.*, using an all-in-one switching ASIC for the data plane. The all-in-one Ethernet switching ASICs, from the companies such as Broadcom, Intel and Marvell, greatly simplify the data plane design. They perform everything including packet parsing, lookup, buffering, scheduling and modification, and all lookup tables and packet buffer are embedded in the ASICs. Second, switch vendors such as Arista and Force10 now use multi-core x86 CPUs and attach gigabytes DRAM in their commodity switches for handling complex control and management plane functionalities, *e.g.*, Arista 7050S series use an AMD 1.5GHz dual core x86 CPU plus 4G DRAM.

While the first design trend has greatly simplified the data plane design of the switches, using these all-in-one ASICs also brings two major limitations to the data plane. First, these ASICs have limited number of forwarding entries for flow-based forwarding schemes such as Openflow. For example, the state-of-art Broadcom switching ASIC has only 2K TCAM entries to match TCP/IP 5-tuple flows. Such limitation makes it very challenging to support this fine-grained forwarding scheme which gives lots of routing flexibilities but requires large forwarding table in return. Second, the on-chip packet buffer of these ASICs is fundamentally limited to several megabytes by the chip die size and can be easily overflowed by the traffic bursts, such as TCP incast [4], TCP flash crowd and low-rate bursty attack traffic [5], resulting in severely degraded TCP performance.

On the other side, as CPU is becoming more and more powerful in packet processing [6, 7], it seems natural to go further along the second trend by augmenting the CPU in these commodity switches and letting it do traffic co-processing, *i.e.*, letting the CPU not only do control plane but also participate in some data plane functionalities. While the major portion of traffic are still processed by the switching ASICs, a small portion of traffic can now be processed by the CPU.

Such design can address the previous two limitations nicely. First, we can now put the whole forwarding table in software and use the switching ASIC to offload traffic forwarding. Since the traffic flows are widely known to be classified as elephants and mice [8], we can *offload* the elephants to the ASIC and *onload* the mice to the CPU, *i.e.*, let the CPU forward the mice. Second, since the DRAM for a CPU can be orders of magnitude larger than the on-chip packet buffer, we can direct the traffic bursts to CPU and use DRAM to buffer them. Furthermore, once the traffic are processed by the CPU, we can program the CPU to perform certain tasks which are difficult for the ASICs. For example, we can detect malicious traffic bursts and mitigate their effects.

We evaluate our design by prototyping such system on ServerSwitch platform [9]. First, our prototype delivers 3.9Gb/s software forwarding throughput for 100k flows and achieves over 90% traffic offload ratio in our testbed setup. Second, our system absorbs large temporary TCP traffic bursts perfectly without dropping a single packet. Last, we can easily program our system to detect and defend distributed low-rate traffic burst attacks.

The paper is organized as follows. We first elaborate our design goals in the § 2 and describe the design in § 3. Then we present the implementation and evaluation results in § 4. We discuss related work in § 5 and finally conclude in § 6.

2. DESIGN GOALS

First of all, we would like to follow the merchant switching ASIC trend and adhere ourselves to this single switching ASIC approach since it has greatly simplified the design of current commodity switches. However, we would like to propose some modifications to current commodity switch design to address its two major limitations. Specifically, we have following two design goals.

Large forwarding table for flow-based forwarding scheme: Flow-based forwarding schemes such as Openflow provide very flexible routing control for better network security [10], high network utilization [11], and power savings [12] in the enterprise network and DCN. On the other side, such fine-grained forwarding schemes require very large forwarding table. For example, at its finest granularity, OpenFlow uses one forwarding entry to match one TCP/UDP flow using its exact match rule. Our measurement results (§3.1) show that the number of active flows passing through a switch can exceed 10K entries (with 60-second timeout), the requirement for the forwarding table size could be very large.

Deep packet buffer for temporary traffic burst absorbing: When the switch packet buffer size is limited, bursty traffic pattern such as TCP incast and a flash crowd of TCP short flows can easily lead to packet drops and cause throughput degradation. In TCP incast, tens of senders simultaneously send traffic to a receiver. When all the traffic arrive at a switch and the switch does not have enough buffer to hold them, packets are dropped which could cause TCP timeouts and significantly enlarge the flow completion times. On the other hand, long TCP flows always saturate the switch buffer to cause packet drops. In a DCN environment where the bandwidth-delay-product (BDP) is small, deep buffer for long flows is not desirable since it does not increase their throughput but only leads to extra queueing delays. Therefore, our goal here is to equip the commodity switches with a deep packet buffer to absorb temporary traffic burst *only.*

Sometimes, the traffic bursts can be malicious. In a multi-tenant data center, it is highly possible that malicious tenants can initiate distributed low-rate burst attacks [5] to dramatically degrade the TCP throughput of other tenants. In this attack, multiple malicious hosts send low-rate bursts to a targeted switch to overflow its buffer. In response, the data center operators should be able to detect such attacks and find the attackers. Since the attack is distributed, the most straightforward solution is to do traffic monitoring and analysis on the targeted switch. Therefore, our subgoal here is to detect these malicious traffic bursts and mitigate their effects.

Unfortunately, current all-in-one switching ASIC cannot meet these two design goals. They only have around several thousands of flow entries [13] and several megabytes packet buffer. There are some ASIC-based solutions. Some switching ASICs can have external lookup tables and packet buffer, *e.g.*, Broadcom switching ASIC BCM56440 can attach external DRAM to provide deep packet buffer. However, such kind of chips is only available for certain product lines. To our best knowledge, for those 10G switching ASICs which are widely used in 10G ToR switches, no such chip is available in the market now or in the near future. Besides, since ASICs have limited programmability, it will be difficult to program them to detect burst attack traffic.

3. DESIGN

In our design, we equip the commodity switches with a powerful CPU and setup a high bandwidth internal link between the CPU and the switching ASIC. We can program the ASIC to direct a portion of traffic from the ASIC to the CPU via the internal link. The CPU then processes these traffic and sends them back to the ASIC afterwards.

Such design can achieve previous two goals. As for the first goal, since the CPU has the complete forwarding table, we can direct the packets to CPU for table lookup when they miss in on-chip forwarding table. As for the second goal, when traffic burst arrives and the queue in the ASIC approaches its limit, we immediately direct the subsequent traffic to CPU and use the DRAM to buffer them temporarily. Meanwhile, we also detect long flows and use ASIC to forward them directly. Once we redirect the traffic burst to CPU, we can further program the CPU to detect the low-rate burst attack traffic. Compared with the ASIC-based solution, this design can be applied to any existing switching ASICs and can introduce more programmability in the data plane easily.

Although CPU can now forward tens of Gb/s traffic, it is still an order of magnitude lower than the switching capacity of merchandised switching ASICs, *e.g.*, 100Gb/s packet processing rate for crystal forest platform v.s. 1.28Tb/s switching capacity for Broadcom Trident2 ASIC. However, fortunately our CPU-based traffic co-processor does not have to process all the data plane traffic to achieve our goals. In the first case, the ASIC already has thousands of forwarding entries or even more. As the traffic pattern is widely known to be classified as elephants and mice, hopefully, we can offload the elephant flows to the ASIC while leaving the mice flows to be handled by the CPU. As for the second case, only temporarily bursty traffic are processed by the CPU.

In the following sections, we will discuss how our design achieves previous two design goals in detail.

3.1 Large forwarding table

We first show the flow statistics measured from a private data center which is primarily used for running MapReduce-style applications. The data center has 120 racks and around 5k servers. We instrumented all servers on 118 racks to record flow data. We uniquely define a flow using TCP/IP 5-tuple. Similarly to previous measurements studies [14], we use a long inactivity timeout of 60 seconds. However, this is due to the limitation of our current instrumentation. We plan to enable much shorter timeout value to have more accurate flow counting in the future. We gathered the flow data for two weeks starting from Feb 2, 2012.

We measured the number of active flows and calculated traffic offloading ratio (TFOR) for each rack. Fig. 1 and 2 shows the statistics of these two metrics for all racks. Let's consider commonly used 10G switching ASICs Broadcom Trident+ and its successor Trident2 which support maximum 2048 and 4096 TCP/IP 5-tuple flow entries. As shown in Fig. 1, the number of active flows is more than 2048 for 99% of the time, and 4096 for 95% time. TFOR is the ratio of the offloaded traffic to the total traffic. To study the feasibility of using CPU to forward a portion of traffic, we divided the flows into one-second intervals. For every second, we rank all active flows based on their bytes transfered in that second and count largest k flows as offloaded traffic. As we can see from Fig. 2, the TFOR is more than 89% for 90% of the time when $k = 2048$, and 92% for 95% of the time when $k = 4096$. Since the forwarding throughput of switching ASIC is about 10 times larger than that of a CPU, it is reasonable to assume that the CPU only forwards less than 10% of the total traffic. The figure suggests that we can potentially offload more than 90% of traffic for 90% of time even when the hardware can only support a fraction of active flows, the hybrid approach makes a lot of sense. As for the rest $5 \sim 10\%$ time, we further find that the maximum onloaded traffic rate is below 4Gb/s which can be easily handled by current CPU. One may ask what happens if the onloaded traffic rate is too large and is indeed bottlenecked by the CPU. In that case, since TCP traffic is elastic, unless the all k offloaded traffic are bottlenecked somewhere else, they alone can saturate the outgoing pipes. As a result, some of them may run faster. Once an offloaded flow finishes, we can immediately offload another onloaded traffic. As long as we can saturate the outgoing pipes, we will not lose overall throughput in long run.

We note that the above measurements only represent a specific type of data centers. The exact number of active flows and TFOR may differ in different data centers, *e.g.*, a virtualized data center may have much more active flows since there could be hundreds of VMs under one ToR switch. However, the above measurements do confirm our common belief that the number of active flows can be much larger than the flow table size in the state-of-art switching ASICs, and that majority of traffic volume are carried by a small portion of flows.

In our design, each flow i offloaded to the ASIC corresponds to a rule R_{flow_i} within the ASIC. When a new flow arrives, we first deliver the flow to CPU. The CPU then checks if there is any inactive flows in the ASIC. Here we define a flow to be inactive if it does not transmit any packets within $T_{inactive}$ period. Since the RTT in DCN is small, we let $T_{inactive}$ to be 1 second. If yes, we swap the inactive flow with the new flow. If there is no inactive flow, the CPU will forward the new flow temporarily.

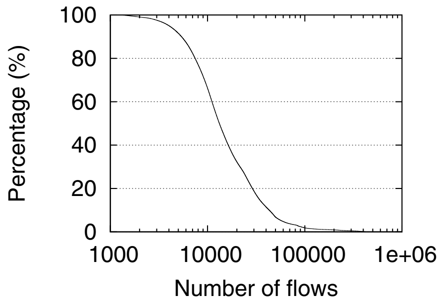

Figure 1: Complementary CDF for number of flows

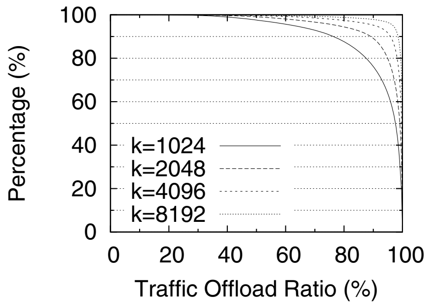

Figure 2: CCDF for traffic offload ratio

To minimize the traffic volume forwarded by CPU, we need to put the flows with larger rates into the ASIC. Suppose the ASIC has k entries, the ASIC then needs to cover the flows with the k largest traffic rates. As the flow rates change dynamically, the on-chip forwarding table needs to be adjusted accordingly. Thus, we periodically obtain the rates for all flows, rank them based on their rates, and finally swap the k largest flows into the ASIC. As for the flows forwarded by CPU, we can easily count the their bytes to get their rates. As for the flows forwarded by the ASIC, it is common that each TCAM flow entry is associated with a byte counter in existing ASICs. Thus, we can read those counters to calculate the flow rates. Currently, we set this ranking interval to be 100 milliseconds.

3.2 Deep packet buffer

Our design uses the DRAM as the off-chip packet buffer to achieve following three goals. First, we absorb temporary traffic bursts such as TCP incast traffic and a flash crowd of TCP short flows. Second, the long flows are forwarded by the ASIC directly. Third, we detect and defend low-rate burst attack traffic. In the next, we'll describe these three designs one-by-one.

3.2.1 Absorb temporary traffic bursts

Fig. 3 shows how the deep buffer absorbs traffic bursts. Here, we define traffic bursts to be any traffic pattern that can overflow the output queue. When traffic bursts come, CPU first setups a rule on the switching ASIC to redirect the bursts to CPU via the high bandwidth internal link,

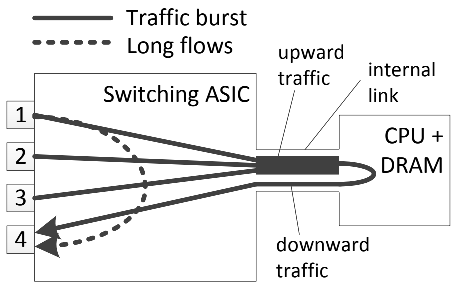

Figure 3: Using deep buffer to absorb traffic bursts

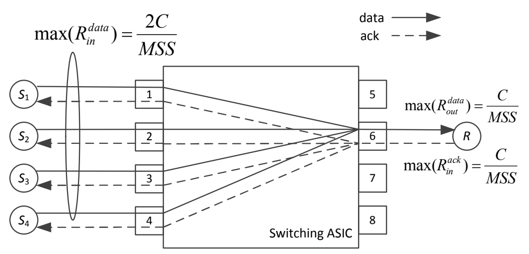
$\max(R_{in}^{data}) = \frac{2C}{MSS}$
data
ack
S_1
S_2
S_3
S_4
1
2
3
4
5
6
7
8
$\max(R_{out}^{data}) = \frac{C}{MSS}$
R
$\max(R_{in}^{ack}) = \frac{C}{MSS}$
Switching ASIC

Figure 4: Maximum incoming rate for TCP burst traffic to one port

and uses DRAM to buffer them. Then, CPU paces these packets to the output port without overflowing the output queue on the switching ASIC. In the following paragraphs, we'll describe how to avoid packet drop in the above steps.

First, CPU monitors output queue length on every link. Once an output queue length exceeds the redirection threshold $Q_{redirect}$, CPU setups a rule R_{port} in the ASIC to redirect all the traffic which are destined to that output port to the CPU. Practically, the rule setup time is usually tens of microseconds. The incoming packets still accumulate in the output port before the rule has been setup. Thus, we reserve some headroom between $Q_{redirect}$ and the queue limit Q_{limit}, usually tens of packets, so that there will be no drop during the rule setup period. When the burst finishes and the DRAM is emptied, CPU then cancels the rule R_{port}. To co-exist with large forwarding table in previous section, the priority of R_{port} is higher than that of R_{flow_i} setup by the large forwarding table so that all the traffic will be redirected once there is burst.

Second, we need to determine the minimum internal bandwidth in order to move the traffic burst to CPU without dropping any packet. Let's first consider the traffic bursts such as TCP incast and TCP flash crowd which are entirely composed of TCP traffic. Consider the case when data traffic all go to a same output port as shown in Fig. 4. Since the maximum outgoing data rate is limited to the port speed C, the maximum data packet rate on the output port (R_{out}^{data}) is C/MSS, where MSS is the maximum segment size. The maximum of ack packet rate on the reverse direction (R_{in}^{ack}) is also C/MSS when delay ack is disabled at the receiver. In TCP slow start phase, the senders increase their congestion windows by one MSS for one received ack. Thus, each ack will trigger one more data packet. The maximum incoming data packet rate (R_{in}^{data}) and data rate are $2 \times C/MSS$ and $2 \times C$, respectively.

Above analysis suggests that once the internal bandwidth is larger than $2 \times C$, this design can fully absorb the traffic burst that happens on one output. In practice, the internal bandwidth can be provisioned several times larger than $2 \times C$ in order to absorb bursts on several output links simultaneously. As traffic bursts are usually on small time scale, *e.g.*, millisecond level, it is unlikely that many bursts collide at this time granularity. For the non-TCP bursty traffic, we only give them maximum $2 \times C$ upward bandwidth because we believe that no traffic should be more bursty than TCP slow start.

Third, CPU caps the sending rate on the downward path to the bandwidth of the output port. When there are no other competing traffic, the sending rate matches the port speed so that the output queue will not build up. No packet will be dropped.

3.2.2 *Forward long flows directly*

In practice, there could be long flows on the same output port of the traffic burst as shown in Fig. 3. The long flows bring two issues. First, since TCP tries to saturate the link, long flows will trigger the $Q_{redirect}$ threshold and will be forwarded by CPU until they end. To solve this problem, we need to treat long flows differently. CPU counts the bytes for every flow. Once a flow j exceeds certain bytes, *e.g.*, 50KB, CPU inserts a high priority rule R_{lflow_j} in the ASIC which overrides the rule R_{port}. The rule also classifies the packets into a specific type and the output queue length for that packet type is limited to Q_{lflows}, where $Q_{lflows} < Q_{redirect}$. Once the rules are setup, the long flows will be forwarded by the ASIC directly and will no longer trigger traffic redirection. Second, the long flows compete with the traffic burst from CPU for the output port, and may cause the traffic burst to drop packets on its downward path. To solve this problem, we put the traffic from the CPU into high priority queue so that the ASIC will not drop them when they are competing with the long flows.

3.2.3 *Defend low-rate burst attacks*

Low-rate burst attack traffic try to saturate the switch queue to generate packet drops. Similarly, we also redirect the bursts to CPU when the queue length exceeds $Q_{redirect}$. Once the bursts go to CPU, we program the CPU to detect if the bursts are low-rate attacks or not. Once CPU has detected the suspicious traffic, it uses a rate-shaper to limit the outgoing rate of the suspicious traffic to BW_{attack} and thus defend normal TCP traffic from the attacks.

We use a very simple detection algorithm, since the algorithm is not the major focus of this paper. More sophisticated detection algorithm [15] can also be implemented in our design. Our algorithm is as follows. We measure the incoming traffic rate to a port at 1ms granularity. When the traffic rate is larger than the output port speed, we treat that period as a busy period. During a certain time interval, *e.g.*, 3 seconds, if the busy periods contain more than 90% traffic volume, are less than 10% time span, and the cycle of busy periods is around minimum TCP RTO, *e.g.*, 300ms for Windows TCP stack, we classify the burst traffic as low-rate burst attacks.

4. IMPLEMENTATION AND EVALUATION

We build a prototype switch with 16 GbE ports on the ServerSwitch platform. We install four ServerSwitch cards into a HP z800 workstation (8 CPU cores and 48GB DRAM), and connect them via their 10GbE XAUI ports to form a non-blocking switching fabric with 16 GbE ports. We activate four internal GbE links between the switching ASICs and the CPU, and thus the internal link bandwidth is four Gb/s. We connect 16 servers to our prototype switch.

We implemented large flow-based forwarding table and deep packet buffer on the ServerSwitch platform. The implementation consists of both userspace and kernel code. The userspace code is for switching ASIC management such as flow insertion/deletion and queue length monitoring. The kernel code is mainly for packet forwarding.

Large flow-based forwarding table. Since part of the TCAM table is reserved for other uses, we have 1,792 TCP/IP 5-tuple flow entries for this experiment. We first benchmark the ASIC flow insertion, deletion and counter read throughput. The ASIC can do 14,144, 9,524 and 86,956 operations per second respectively. As we will see in our 2nd experiment, these numbers are enough for our dynamic flow management.

In the first experiment, we generate 50k bidirectional (100k uni-directional) TCP flows among four servers. We manually configure all traffic to be forwarded by the CPU. As we measured, these 100k flows fully saturate the 4GbE internal links and the total forwarding throughput is 3.9Gb/s with 14% CPU utilization (about 1 core). In the 2nd experiment, we first get the average flow interval and extract empirical flow size distribution from our previous DC measurements, and then generate synthesized flows based on the interval and the distribution among 8 servers. We run the experiment for 10 minutes. It generates 103,200 TCP flows and 33.6GB data. The maximum number of active flows is 10,644 and the TFOR is 96.1%. To further test our design, we compress the average flow inter-arrival time to its 1/10 and re-run the experiment. The maximum number of active flows is now 106,544 and the TFOR is 90.5%. In average, there are 1,743 flow swaps per second in the latter test which is far below the ASIC capability.

Deep packet buffer for traffic burst absorption. We show how deep packet buffer can absorb temporary traffic bursts caused by a flash crowd of TCP short flows. This traffic pattern is to emulate a surge of requests sent to a web server cluster while all the responses go through a bottleneck link. We use one server to generate all the requests and have the other 15 servers acting as the web servers. All the responses go to the 1GbE bottleneck link to the client. All responses are 256KB. The client continuously generate requests for one second. We vary the number of requests (NR) from 128 to 1,024 by tuning the interval between two requests. When $NR = 1,024$, the response throughput is 140Mb/s per server and the total throughput is 2Gb/s. We test three experiment setups, *i.e.*, TCP with deep buffer, TCP w/o deep buffer, and DCTCP w/o deep buffer. For all setups, Q_{limit} is set to 100 full-sized Ethernet packets, $Q_{redirect}$ is 80 packets, and the ECN marking threshold is 20 packets.

Fig. 5 shows the 99 percentile of response finish times (RFT). As we can see, RFT grows almost linearly when deep buffer is used. This is because there is no single packet drop in this case and the RFT growth is purely due to the queue build-up. To note, when $NR = 1,024$, deep buffer uses maximum 16MB DRAM for packet buffering which is much smaller than the total DRAM of the server. On the contrary, a portion of flows experience very large RFT, *i.e.*, 3 seconds, when deep buffer is not used. This is because

	SYNACK TO	Data TO	Pkt drops	Avg. RFT (s)
TCP with DeepBuf	0	0	0	0.65
TCP w/o DeepBuf	109	180	15962	0.82
DCTCP w/o DeepBuf	23	395	3302	0.78

Table 1: Packet losses, TCP timeouts and average response finish time when $NR = 1024$

that there are lots of packet drops and TCP timeouts, especially the SYNACK timeouts (3 seconds). Table 1 compares the packet losses and average RFT for $NR = 1024$. As we can see, the average RFT for deep buffer is about 20% less than the other two cases. TCP has much more SYNACK timeouts than DCTCP because the queue is almost full when ECN is not used. On the other hand, DCTCP has more data packet timeouts than TCP even when DCTCP has much less packet drops than TCP. This is because the congestion windows of DCTCP is very small due to received ECN signals. Thus, the packet drops are more prone to turn into timeouts due to small number of in-flight packets.

TCP long flow performance. We show how our deep buffer affects long TCP flows. We setup n long flows from two senders to one receiver for 10 seconds, where the n is set to 2, 4, 8, 16, 32 and 64. Q_{lflows} is set to 50 packets. We also do the same experiments with deep buffer disabled. The aggregate throughputs for both cases can saturate the 1GbE link with 949 Mb/s. When deep buffer is used, we observe 315 out-of-order arrivals (early arrivals) for all 126 flows when the flows are switched from CPU to the ASIC (one switch per flow). The out-of-order arrivals only trigger 22 TCP fast recovery. This experiment shows that the deep buffer has almost no effect on the TCP long flow throughput.

Defend low rate burst attacks. We setup one TCP connection between one sender and one receiver, and let the other 14 servers initiate low rate burst attacks to the receiver. Each attacker sends 200KB UDP bursts at 200Mb/s peak rate. The interval between two bursts is 320ms. The average throughput for each attacker is only 5.12Mb/s. All attackers are synchronized. When deep buffer is used, we detect and defend the low-rate attacks using the algorithm in § 3.2.3. The BW_{attack} is set to 200Mb/s.

Fig. 6 shows the average TCP throughput in 100 seconds. When deep buffer is not used, the TCP throughput decreases to 40Mb/s when there are 14 attackers. When deep buffer is used, the TCP throughput only decreases by around 70Mb/s, which is the average throughput of all attack traffic. It shows that deep buffer can effectively protect the TCP connection from the low rate burst attacks. Besides, our kernel code for the algorithm plus the rate limiter takes no more than 1,000 LoC which is very easy to implement.

5. RELATED WORK

OpenFlow (OF) defines a centralized architecture to manage the switches. Our design can provide better support for OF in terms of large flow forwarding table. However, the local CPU on the switches are dummy components in the OF architecture, whereas it is much more intelligent in our design as it swaps the flows, monitors the queues and decides the traffic redirection. Our design does require a local CPU to meet the needs of low-latency signalling and high

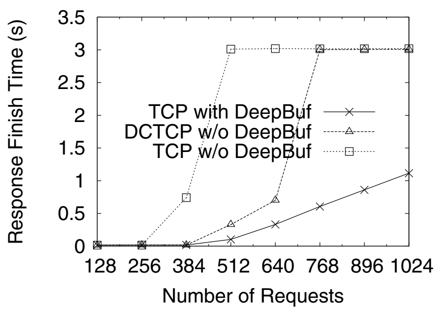

Figure 5: Using deep buffer to Absorb TCP flash crowd

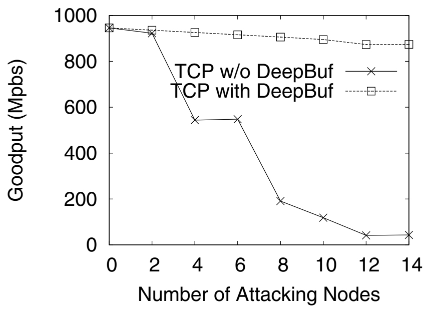

Figure 6: Defend low-rate burst attacks

throughput data transfer between the switching ASIC and the CPU.

DevoFlow [13] also shows that the hardware limitations, such as flow table size, table update rate and statistics gathering rate, can limit the scalability of OpenFlow. The paper proposes several in-ASIC enhancements to address these problems as their design goal is to keep flows in the data plane as much as possible. We take a different approach to address these limitations by using software to forward large number of short flows, thus the hardware does not have to have large flow table. We view these two approaches are complementary to each other. A switch can adopt both approaches to enable fine-grained forwarding scheme.

Nadi Sarrar et al propose a traffic offloading system which uses both software and hardware for packet forwarding in the Internet [8]. Our large forwarding table idea is similar to theirs. However, we focus on enabling flow-based forwarding scheme in DCN environment whereas they focus on IPv4 forwarding scheme in the Internet environment. More importantly, we propose to redesign the commodity switches by using CPU as a traffic co-processor. Our design lets CPU play a more generic role on the data plane than the previous work, *e.g.*, absorbing temporary traffic bursts.

6. CONCLUSION AND FUTURE WORK

Single switching ASIC design have greatly simplified the design of commodity switches. In this paper, we propose to use CPU as a traffic co-processor in these switches to address the two major limitations of this design *i.e.,* the limited forwarding table size and shallow packet buffer. Our initial experimental results show that we have achieved our design goals. Besides, putting CPU into the data plane makes the network devices even more programmable. In longer term, we believe that building SDN upon these more programmable nodes can offer more network functions.

While our initial results are encouraging, there are still many questions to be addressed. In the future, we plan to study how to protect the control plane unaffected since the CPU are now occupied for the data plane. We will also investigate the CPU forwarding latency and out-of-order packets during the path switching phases, and understand how they affect both short and long flows. Very recently, OpenvSwitch [16] has adopted a similar idea of processing short flows in user level and other flows in the kernel. We also plan to compare this with our system. Our initial design begins with single chip switches such as ToR switches and it would be interesting to see if such design can also be applied to multi-chip switches such as the aggregate or core switches.

7. REFERENCES

[1] R. N. Mysore *et al.*, "PortLand: a Scalable Fault-tolerant Layer 2 Data Center Network Fabric," in *Proc. of ACM SIGCOMM*, 2009.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: a Scalable and Flexible Data Center Network," in *Proc. of ACM SIGCOMM*, 2009.

[3] "Lippis Report 182: Top 10 Findings: The Cloud Network Industry Test of 10/40GbE Fabrics." `http://tinyurl.com/789t67a`.

[4] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proc. of ACM SIGCOMM*, 2009.

[5] A. Kuzmanovic and E. W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants)," in *Proc. of ACM SIGCOMM*, 2003.

[6] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism to Scale Software Routers," in *Proc. of ACM SOSP*, 2009.

[7] "Intel's Next-Generation Communications Platform Key to Accelerated Network Services." `http://tinyurl.com/84kwth4`.

[8] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's Law for Traffic Offloading," *ACM CCR*, Jan 2012.

[9] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "ServerSwitch: A Programmable and High Performance Platform for Data Center Networks," in *Proc. of USENIX NSDI*, 2011.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proc. of ACM SIGCOMM*, 2006.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of USENIX NSDI*, 2010.

[12] B. Heller *et al.*, "ElasticTree: Saving Energy in Data Center Networks," in *Proc. of USENIX NSDI*, 2010.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-Performance Networks," in *Proc. of ACM SIGCOMM*, 2011.

[14] T. Benson, A. Akella, and D. A. Maltz, "Network Trafffic Characteristics of Data Centers in the Wild," in *Proc. of ACM IMC*, 2010.

[15] H. Sun, J. C. Lui, and D. K. Yau, "Defending against low-rate tcp attacks: Dynamic detection and protection," in *Proc. of IEEE ICNP*, 2004.

[16] `http://openvswitch.org`.