Rethinking the Design of OpenFlow Switch Counters

Ji Yang
Xi'an Jiaotong University

Chengchen Hu
Xi'an Jiaotong University

Peng Zheng
Xi'an Jiaotong University

Ruilong Wang
Xi'an Jiaotong University

Peng Zhang
Xi'an Jiaotong University

Xiaohong Guan
Xi'an Jiaotong University
Tsinghua University

CCS Concepts

•**Networks** → **Network design principles;**

Keywords

OpenFlow, SDN counter

1. INTRODUCTION

Software Defined Networking (SDN) offers a flexible network architecture by decoupling the control plane and data plane. As the *de facto* implementation of SDN, OpenFlow uses a flow-based control abstraction: switches forward packets based on the fine-grained flow entries installed by the controller. Indicated in [1], the flow table related fast memory accounts for more than 70% of the chip area and 50% of the power consumption in the data path processing silicon.

In addition, the latest OpenFlow specification 1.5.1 [5] defines a bunch of counters for each port and type of tables to track network status and enforce control functions. If all these counters are set in the OFS processing ASIC, an extra 49.6% memory will be required, which makes implementation infeasible or extremely expensive. In most of the cases, the counters are sarificed when competing with flow tables.

Besides the OFS fast path, *i.e.*, OFS ASIC, a slow path in OFS implemented by a CPU handles the infrequent coordination with the controller and the configuration to the fast path. We observe that the cache size in the up-to-date CPU could reach 512KB-2MB and the processing load in slow path CPU is usually light. As a result, we start rethink the OFS architecture to move the counter array originally in the fast path to the unused CPU cache in the slow path.

As the first contribution of this work, we propose CAche CounTIng (CACTI) to fulfill the counting tasks with almost-zero memory consumption in the fast path of OFS. Specifically, CACTI reserves only several register in the fast path. The second contribution is real implementation of CACTI prototype systems on two hardware platforms. One is all-progrmmable switch (ONetSwitch [4]) and the other is a FPGA-based PCI Express Card plugged into a x86 server.

The results on real traces demonstrated that CACTI can process up to 39.7M packets per second or 316 Gbps throughput with less than 0.42% relative error. The cost in the fast path accounts for about 0.24-0.54% Look-Up Table and 0.35-0.43% flip-flops compared with the whole OFS design. Meanwhile the counter array kept in the slow path can be greatly compressed to fit into the size of CPU cache.

2. DESIGN

The overall architecture of CACTI is shown in Figure 1. A counter update request from OpenFlow datapath consists of the counter identification (Counter ID) and the value to be updated (Updated Value), The *Counter ID Register(s)* records the Counter ID of the most-recently update request, and the *Shadow Counter Register Update Logic* compares the Counter ID of the current counter update with what is recorded by the Counter ID Register(s). If they are the same, the updated value will be accumulated to the *Shadow Counter Register* in the *Shadow Counter Register Array*; otherwise, the value stored in shadow counter register will be sent to the slow path through Direct Memory Access (DMA), and the new update value will overwrite the previous one.

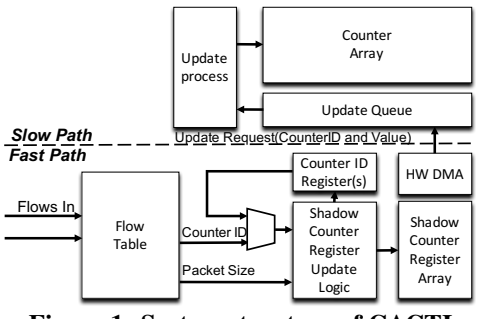

Figure 1: System structure of CACTI

This paper is supported in part by the National Natural Science Foundation of China (61221063, 61272459, U1301254), 863 High Tech Development Plan (2012AA011003) 111 International Collaboration Program of China, Program for New Century Excellent Talents in University (NCET-13-0450) and the Fundamental Research Funds for the Central Universities.

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil
 ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2959062

In the slow path, a *Counter Array* keeps counters for all flow entries of the switch, and it is accommodated into the internal cache of the switch's CPU. The *Compression* module reads update values from the fast path, and accumulate them to the corresponding entries in the *Counter Array*.

The first challenge is how to compress the increment value in order to store all the counters in CPU cache instead of an extended memory. Let n be the flow size, and a compression method should map it into a counter value c that uses less bits than n. Define $f(c)$ as the estimated flow size given a counter c, and the compression function is then $f^{-1}(\cdot)$. Here, we choose the compression method in DISCO [3] which achieves an unbiased estimate $f(c) = \frac{b^c-1}{b-1},\ (b > 1)$. Since there is no enough memory size to maintain floating-point number counters, we could only rely on the integer counters. We can prove that the value stored in CACTI is an unbiased estimate of the counter value if $f(c)$ is an unbiased estimate of the counter value. Note here besides DISCO, any other SRAM-based counter compressing method could be used for CACTI.

A second challenge is how to deal with the unmatched processing speeds of the fast path and the slow path. For example, when there is a burst of small packets, the fast path may overwhelm the CPU in the slow path with a large number of counter update requests. To address this problem, we let the slow path maintain an update queue to keep all the counter update requests.

Optimization: In the above, we assume there is only one shadow counter register, while we need to consider scheduling strategies when there are multiple shadow counters registers. Least Recently Used (LRU) strategy updates the least recently used shadow counter register to the slow path, expecting it keeping an inactive flow. Flush is a greedy algorithm, which updates all counters in shadow counter register array to CPU when no free shadow counter can serve the arriving packet. The target of this algorithm is to optimize the usage of local bus between the fast path and the slow path.

Since DISCO needs 15 bits to store a compressed 64bit counter, CACTI uses the lower 15 bits in a counter to store data and uses the highest 1 bit as the compression mark. Only counters larger than 2^{15} will be compressed and marked.

3. IMPLEMENTATION AND EVALUATE

We prototyped CACTI with ONetSwitch, as well as a x86 platform carrying a FPGA datapath. Since ONetSwitch contains an ARM processor in its slow path, we stored the counter array at on-chip memory. For the x86 platform, we use precompile command to let CPU load the counter arrays before using. For inputs, we synthesize three traces (UNI, EXP, and PAR) based on different statistic parameters and four real traces (CA1-4) from CAIDA anonymous high-speed backbone between 2008 and 2014. During the experiment, we use 16 shadow counters with flush strategy for CACTI. The processing speed and throughput are given in Table 1.

We analyze the relative error of CACTI, DISCO and ICE-buckets [2] under different trace scenarios. CACTI is always better than DISCO, which agrees with the theorem. With different flow distribution, CACTI reduces the upper bound of counting error by 12%. CACTI shows much better accuracy with all traces, especially under synthetic traces which have less small flows. The counter value is compressed when the full size counting overflows.

Table 1: Performance of CACTI on platforms

Scenario	UNI	EXP	PAR	CA1	CA2	CA3	CA4
Processing Speed at x86(Mpps)							
DISCO	25.0	25.0	25.2	20.3	21	20	18.6
CACTI	33.5	33.5	29.4	37.5	39.7	34	30.6
Processing Speed at ONetSwitch (Mpps)							
CACTI	2.53	2.55	2.56	2.60	2.60	2.59	2.58
Throughput at x86 (Gbps)							
DISCO	157	60.8	92	160	170	161	175
CACTI	210	81.1	122	307	316	273	190
Throughput at ONetSwitch (Gbps)							
CACTI	16.0	15.5	17.0	21.5	19.3	20.2	19.6
Update Traffic Reduce Percentage (%)							
CACTI	86	86	86	85	85	82	71

The reduced traffic results are depicted in Table 1. Basically, the performance improves when the number of shadow counters increases, but the margin decreases. Considering the overhead of using more shadow counters, 16 shadow counters should be enough. These results indicate that CACTI significantly reduces the update requests between fast path and slow path without disturbing normal messages in the slow path such as packet-in, packet-out, and flow-mod.

4. CONCLUSION

This paper presents a new architecture named CACTI to implement OpenFlow switch couters. The proposed CACTI has several advantages: 1) In CACTI, the memory cost related to OFS counters is removed from the fast path to the slow path, as a result, which can be saved for large flow tables. 2) It significantly reduces the counter update traffic between the fast path and the slow path, and therefore provides large throughput for counting high speed network traffic. 3) The accuracy of CACTI is no less than SRAM-based counting solutions even in the worst case, and usually shows more improvements on real traces. 4) More optional counters can be fit into OFS by using CACTI to support different controller applications. 5) The implementation can easily be implemented with different hardware platforms.

5. REFERENCES

[1] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM CCR*, 2013.

[2] G. Einziger, B. Fellman, and Y. Kassner. Independent counter estimation buckets. In *IEEE INFOCOM*, 2015.

[3] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, C. Wu, and Y. Cheng. DISCO: Memory efficient and accurate flow statistics for network measurement. In *IEEE ICDCS*, 2010.

[4] C. Hu, J. Yang, H. Zhao, and J. Lu. Design of all programmable innovation platform for software defined networking. In *Open Networking Summit*, 2014.

[5] ONF. OpenFlow Switch Specification Ver 1.5.1, 2015.