Design of a hardware architecture for flow statistics with optimal space and time efficiency

17:02, February 28, 2020

The need for high performance per-flow network traffic monitoring with efficient counter architecture when implementing in real networks has arisen for the following two reasons. Firstly, lots of networking operation applications and data streaming control algorithms require a large array of counters to track various network statistics at line rate, causing a software-based solution hard to handle. Secondly, current hardware-based network monitors, especially for programmable-logic-based devices (*e.g.*, FPGAs), have significant limitation in terms of on-chip memory resource. In this Letter, we, for the first time, propose a hardware-based non-linear counting statistic algorithm that consumes much less RAM resource and has a much simpler circuit design for completely avoiding the logarithm calculation. We implemented a prototype on Xilinx ZC706 board and evaluated the performance and resource utilization. The evaluation result suggests the maximal throughput of 133 Gpbs at 200MHz, which is a 12x performance speedup compared with a pure CPU implementation. In terms of resoure utilization, our implementation consumes no more than 0.5% extra logic resoure and 37.5% less memory resource.

Introduction: Passive measurement examines network traffic to collect statistics without affecting the existing network status. This is an essential element of networking, especially in large-scale Data Center Networking (DCN), which is recently powered by the SmartNIC. Since most of data streaming or management algorithms require indispensable counting information for network monitoring, operations, diagnosis, planning, engineering, accounting, security, *etc*.

Nowadays, lots of novel algorithms and applications are built in the Cloud and the bare metal environment, *e.g.*, BBR[1] (congestion control algorithm), hypervisor[3] (VM operator), which pose promiscuous measuring demands. However, implementing a function unit that can meet all measurement needs will lead to significant on-chip resource consumption. On the other hand, re-developing a function unit every time for a new coming network measurement request is not a time efficient way for network operators. There are many applications of passive measurement in the literature which always have limitations, *e.g.*, only supporting a narrow range of application (Lossradar[2], PLANCK[4]), leveraging active probes and sharing a portion of bandwidth (Flowradar[5], Pingmesh[6]). Besides, the ever-increasing link speeds and number of flows lead to significant challenges to the software-based counter architecture. Therefore, it inspires us to propose a hardware-based programmable passive measurement engine for DCNs.

Basic abstraction: The per-flow statistics is the abstraction, which keeps counters for all the flows with low memory overhead and page the counter information to the remote controller for performing network-wide analysis. The key design of the abstraction is to convert per-flow statistics to the target flow matrix. The flow matrix is essential information for networking, *e.g.*, security (ACL, Firewall), traffic congestion control (CC), etc, as Table 1 describing the converting relationship.

Table 1: How to convert flow matrix from per-flow statistics

	flow matching	pacekt loss	tx/rx throughput
ACL (heavy hitter)	✓		
Firewall (large flow)	✓		✓
CC	✓	✓	✓

Existing flow statistics monitor: Within network measurement context, traditional flow statistics *e.g.*, exact counting, is not efficient for SmartNIC (large memory cost). Sampling-based traffic monitor is a promising solution to deal with the huge amount of traffic traversing network devices, since only a subset of packets is elected for analysis. Estimators can represent large counting values with small memory space at the price of a small measurement error. A compression algorithm DISCO[7] improved the estimation accuracy, which is by far the best error bounded sampling function proofed by later work[8].

DISCO overview: Mathematical principle of compressed counting method. We introduce the mathematical principle of an unbiased estimation for compressed counting, named DISCO [1]. As mentioned in *Introduction*, the goal of our statistics method is to compress the counter bits in order to fit the fast but small memory space. Suppose c is the counter value and we use a function $n = f(c)$ to express the total counted flow size (n),

$$f\left(c\right) = \frac{b^c - 1}{b - 1} \tag{1}$$

where b is a predefined constant parameter. It is obvious that $c = f^{-1}(n)$ is an increasing concave function, so the growth of the counter value must be sub-linear. The compression ratio is controlled by parameter b, which makes the counting mechanism scalable. The number c should be an integer in order to use the small counter space efficiently, because storing the decimal number costs more memory size. The counter should be increased by $\Delta(c, l)$ from its previous value c when a packet of l bytes comes, where $\Delta(c, l) = f^{-1}(l + f(c)) - c$. When the counter's value is large enough, c could not increase every time a packet length l is counted. As a result, we increase the counter by $\delta(c, l) + 1$ with probability of $p_d(c, l)$, and increases the counter by $\delta(c, l)$ with probability $1 - p_d(c, l)$, which is defined as

$$\delta(c, l) = \lceil f^{-1}(l + f(c)) - c \rceil - 1 \tag{2}$$

$$p_d(c, l) = \frac{l + f(c) - f(c + \delta(c, l))}{f(c + \delta(c, l) + 1) - f(c + \delta(c, l))} \tag{3}$$

To achieve the counting compression, we have to perform the inversion of two complex functions that involves logarithmic calculation.

However, implementing DISCO on a SmartNIC faces two challenges: 1) Complicated calculation in DISCO. Hardware-based logarithm calculation is inefficient for a SmartNIC. 2) High memory resource consumption. A pre-computed lookup table for calculation has a large memory resource overhead. Suppose we pre-compute each l in range of (64, 1516) for every different value c (0, 3072), we need a memory space of 1516x3072x32bits (about 150Mb), which can not be stored in on-chip fast memory by SmartNIC.

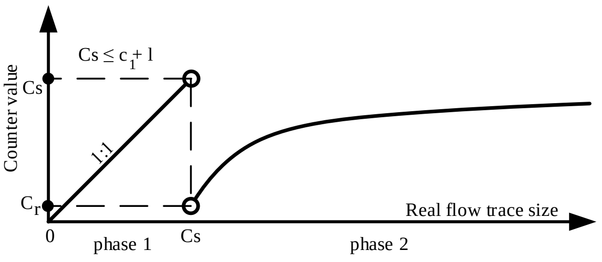

Fig. 1. Intuitive illustration of how a flow counter grows with flow size

We introduce an equivalent calculating solution which is suitable for hardware circuit design. The counter update process for each flow is separated into two phases. As shown in Fig. 1, it starts to count a new flow firstly in phase 1 with initial counter value of 0. In this phase, the exact value (packet length) is recorded precisely by the counter. Increasing counter value in phase 1 will cause an overflow in short time under the heavy network traffic. When the counter value in phase 1 is going to exceed a predefined parameter C_s, the counting compression phase (phase 2) will start. By a careful calculation of C_s, the counter can be mostly increased by 1 (or by 0) with a probability function $p_c(c, l)$ shown in Eqs. (4), which means the $\delta(c, l) = 0$.

$$p\left(c, l\right) = \frac{l}{f\left(c + 1\right) - f\left(c\right)} \tag{4}$$

where the l is the length of an incoming packet (ranging from 64 to 1516), c is the current counter value stored in fast memory. In addition, Assume the counter value in phase 1 is $c_1 (c_1 < C_s)$, if a incoming packet with size l, which makes $c_1 + l > C_s$, as shown in Fig. 1, a counter value re-normalization will be triggered: counter value will be changed from c_1 to C_r. Where C_r can be calculated as

$$Cr = \{ {}^{\lceil f^{-1}(c_1 + l) \rceil,\ with\ probability\ of\ p_r,}_{\lceil f^{-1}(c_1 + l) \rceil - 1,\ with\ probability\ of\ 1 - p_r} \tag{5}$$

$$p_r = \frac{c_1 + l - C_s}{f(f^{-1}(C_s) + 1) - f(f^{-1}(C_s))} \tag{6}$$

where the $f^{-1}(C_s)$ and $f(f^{-1}(C_s)+1)$ can be pre-calculated. There is one flag bit in each flow associated counter, which serves as an indicator of whether it is in the first or second counting phase. This one-bit flag is initialized to zero and is set to one when entering the counting phase 2. It represents the phase of counting update or/and the reverse estimate after checking the one-bit flag.

After the measurement, if the counting stats is only within phase 1, the counter value is the flow statistics result; otherwise, the function $f(c)$ defined in Eqs. (1) is used as the inverse estimate of the flow statistics result for each flow. Parameter C_s is used to determine different measurement phases, and it can be pre-computed it in following way: we firstly find the minimum threshold n_0 defined as

$$\lceil f^{-1}(n_0+l) - f^{-1}(n_0)\rceil = 1, l \in [0, L_{max}] \quad (7)$$

$$C_s = \lceil f^{-1}(n_0)\rceil \quad (8)$$

by a binary search algorithm, where $n_0 \in [0, f(c_{max})]$, $C_s \in [0, 2^x]$ and $L_{max}$is the largest packet size (1516 bytes), the x is the number of bits to accommodate the counter with value C_s. Then we define C_s using Eqs. (8). Notably, b is a predefined parameter that controls the compression ratio and counting error E, defined as

$$E = \mathbb{E}(\frac{|n_{real} - f(c)|}{n_{real}} * 100\%) \quad (9)$$

The compression ratio is defined as the maximum counting value in phase two divided by the maximum counting value with exacting adding like phase 1, which can be computed as $f(2^x)/2^x$. The relative counting error is the ratio of the estimate difference (between real flow bytes and estimate) and the real flow bytes. We use the absolute value of the relative error, which is always positive. When we increase $b(b>1)$, the compression ratio also increases, which induces larger relative error. We illustrate below in Table. 1 and Fig. 2 the compression/error comparison and counter value/counting range with b of different values. As a rule-of-thumb, $1 < b < 1.05$ would be a proper range to have a good trade-off between compression ratio and relative error.

Table 2: How does the setting of b affect performance

b	Compression ratio (R)	Estimate err (E)
1.003	2.63	3.87%
1.004	39.7	4.47%
1.005	637	5.00%
1.006	$1.1 * 10^4$	5.48%
1.007	$1.8 * 10^5$	5.92%
1.008	$3.2 * 10^6$	6.32%
1.009	$5.6 * 10^7$	6.71%
1.010	$9.9 * 10^8$	7.07%
1.050	$3.3 * 10^{68}$	15.80%

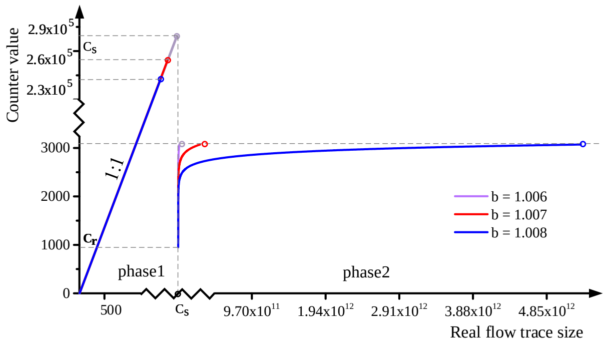

Fig. 2. Different counting curves with different b

Hardware architecture: The hardware architecture design, namely Statistics Measurement Engine (SME), is depicted in Fig. 3. The measurement results are based on flows, each of which is defined by (exact or wildcard) matching of a number of protocol fields. When a packet comes into the SME, the flow classification module will assign a flow-id to it according to a flow classification configuration (via control plane interface) and extract the packet length for measurement purpose. It is worthy of note that the packet length defined as one if counting for flow size (number of packets in a flow) and as the number of bytes of the packet if counting for flow volume (bytes of a flow). A measurement meta-data <*flow-id, packet length*> will trigger the subsequent *scheduler* module, which is connected to the counter update module and counter arrays. There are two blocks of counter arrays that active alternately: at each time, only one of them is active in updating the counting results, while the other one passes all counting results to the control plane, in order to achieve a non-blocking updating progress. At the first step, the scheduler module uses flow-id as the key to query the counter value from the active counter array and fetches the current counter value c. Afterwards, the scheduler sends the counter value c and the packet length l to *counter calculation* module to requrest a new counter value which should be committed to the counter array.

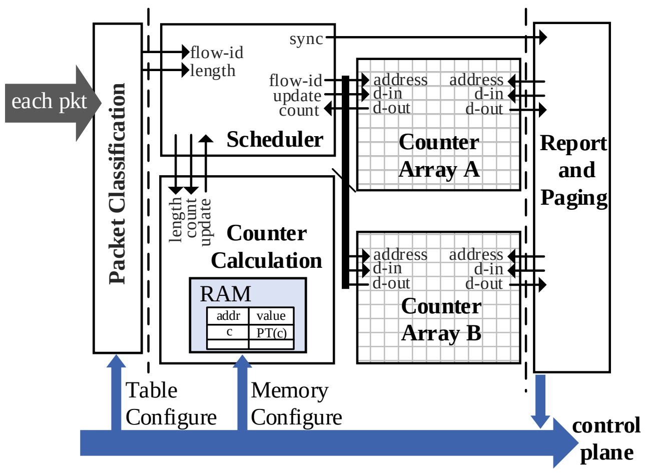

Fig. 3. Architectural diagram of Statistics Measurement Engine (SME)

Pre-computing table in RAM: : In the counter calculation module, we need to calculate the probability $p(c,l)$ so as to decide whether to add one to the current counter value. Instead of directly calculating the Eqs. (2) every time when a packet arrives in phase 2, we use a pre-computed table to reduce the cost of calculation. The pre-computed table keeps the value of $\frac{1}{(f(c+1)-f(c))} * \frac{MAXNUM}{L_{max}}$, where $MAXNUM = 2^x$ is the largest number could be represented by the system. Since $f(c+1)-f(c)$ is between one and $MAXNUM$ (c is an integer), the value of $\frac{1}{(f(c+1)-f(c))} * \frac{MAXNUM}{L_{max}}$ in the pre-computed table, which we denote as $PT(c)$, is also between one and $MAXNUM$. When the counter value in the second counting phase needs to be updated, a random value between zero and $MAXNUM$ is generated: if it is smaller than $PT(c)$, the counter size will be increased by one, otherwise the counter size does not change. Fig. 4 is an example when $b = 1.006, x = 19, width = 20$. Assume that the current counter c is 1254 and the packet length is 512bytes (data width is less than 11bits), get the result of the probability, $p(c,l) = PT(C) * l = 1214400512$. Considering that the $MAXMUN$ is 4294967296, the probability of counter increment is about 28% ($p_c/MAXMUN$). Because Xilinx FPGA can support an ASIC level performance 25 (bits) x 18 (bits) multiplier (our requirement is 19x11), the multiplication can finish within 1 clock cycle. In this way, the counter calculation module can guarantee the result return latency by 2 clock cycles. Besides, the whole statistics progress needs one more RAM inquiry operation and one more RAM update, which may cause performance degradation for processing small-sized packet of 64 Bytes. In this regard, we can place a small number of shared cache (10bits, max range is 1024) for active flows. The cache counts the small length value firstly with no compressing processing, and then a larger cache value will be send to the SME to update the compressed counter array before the cache is about to overflow. In this way, we can achieve a full pipelined processing. In the hardware implementation, the top 1250 space of RAM is unused and can be removed to release more on-chip memory resource.

Results: We perform all our experiments on ZC706 FPGA (900+ lines of Verilog HDL code) development board. A packet generator is integrated in FPGA to generate traffic for testing. The result shows that only one SME engine can handle 133Gbps traffic (64bytes packet length) with 200MHz system clock, which achieves 12x performance speedup compared with a pure CPU implementation [1]. Table. 2 shows the resource consumption of

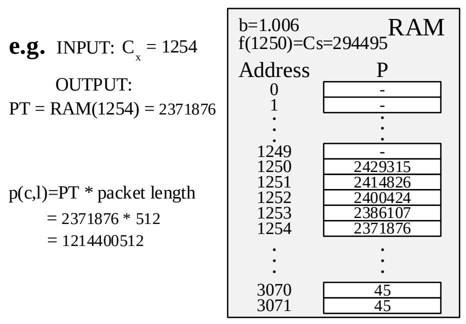

Fig. 4. Different counting curves with different b

Table 3: The implementation result of the statistics measurement engine (SME)

LUT	LUT RAM	FF	SDP	BRAM PT.1k	PT.4k	PT.16k
933	20	599	1	6	12	36
0.43%	0.03%	0.14%	0.11%	1.0%	2.2%	6.6%

*we only show the consumption of core modules and counter arrays except for the classification module, because that we can reuse the existing classification modules in forwarding devices.

the SME implementation: the SME engine only consumes with no more than 0.5% logic resource. For the counter storage, we only use 20bits counter space for each flow. Comparing with the linear 32bits counter, we save 37.5% storage resource, and get a 100x expanding of counting range (when chose $b = 1.007$). When implement $16k$ entries for PT table, the total BRAM consumption is 6.6%.

Conclusion: In order to efficiently measure flow size with high performance on hardware-base network devices, this Letter proposes a calculating-optimized and high-performance counting structure, along with a cost-efficient compression algorithm and a time-efficient hardware-based calculation scheme. The Statistics Measurement Engine (SME) can expand the counting range for a fixed size of counter with an acceptable loss of measurement accuracy and extra logic resource consumption of FPGA. Compared with the state-of-the-art software-based compressed counting method, we improves the throughput by 12 times.

Acknowledgment: This work has been supported by The IET

J. Smith and A. N. Other (*The IET, Stevenage, UK*)

E-mail: jsmith@theiet.org

References

1 Neal Cardwell, et al.: ‘BBR: Congestion-Based Congestion Control’, *ACM queue*, 2017, **60.2**, doi:10.1145/3009824
2 Li, et al.: ‘Lossradar: Fast detection of lost packets in data center networks’, *In Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, **ACM**, pp. 481-495
3 Firestone, et al.: ‘Azure accelerated networking: SmartNICs in the public cloud’, *In 15th USENIX Symposium on Networked Systems Design and Implementation*, , **NSDI 2018**, pp. 51-66
4 Rasley, et al.: ‘Planck: Millisecond-scale monitoring and control for commodity networks’, *In ACM SIGCOMM Computer Communication Review*, 2014, **44, 4**, pp. 407-418
5 Li, et al.: ‘Flowradar: A better netflow for data centers’, *In 13th USENIX Symposium on Networked Systems Design and Implementation*, **NSDI 2016**, pp. 311-324
6 Guo, et al.: ‘Pingmesh: A large-scale system for data center network latency measurement and analysis’, *In ACM SIGCOMM Computer Communication Review*, 2015, **45.4**, pp. 139-152
7 Hu, et al.: ‘Discount counting for fast flow statistics on flow size and flow volume.’, *IEEE/ACM Transactions on Networking* , 2013, **22.3**, p. 970-981
8 Einziger, et al.: ‘ICE Buckets: Improved Counter Estimation for Network Measurement’, *IEEE/ACM Transactions on Networking (TON)*, 2018, **26.3**, pp. 1165-1178