Speed Records in Network Flow Measurement on FPGA

Arish Sateesan*, Jo Vliegen*, Simon Scherrer†, Hsu-Chun Hsiao‡, Adrian Perrig†, and Nele Mentens*§
*imec-COSIC/ES&S, ESAT, KU Leuven, Belgium; Email: {arish.sateesan,jo.vliegen,nele.mentens}@kuleuven.be
†Department of Computer Science, ETH Zurich, Switzerland; Email: {simon.scherrer,adrian.perrig}@inf.ethz.ch
‡National Taiwan University, Taiwan; Email: {hchsiao}@csie.ntu.edu.tw
§LIACS, Leiden University, The Netherlands

***Abstract*—Network traffic measurement keeps track of the amount of traffic sent by each flow in the network. It is a core functionality in applications such as traffic engineering and network intrusion detection. In high-speed networks, it is impossible to keep exact count of the flow traffic, due to limitations with respect to memory and computational speed. Therefore, probabilistic data structures, such as sketches, are used. This paper proposes Approximate Count-Min sketch or A-CM sketch, a novel variant of the Count-Min sketch algorithm that uses less memory and has a higher throughput compared to other FPGA-based sketch implementations. A-CM sketch relies on optimizations at two levels: (1) it uses approximate counters and the newly proposed Hardware-oriented Simple Active Counter algorithm to efficiently implement these counters; (2) it uses a distribution of the embedded memory, optimized towards maximum operating frequency. To the best of our knowledge, A-CM sketch outperforms all other FPGA-based sketch implementations.**

I. Introduction

In many network applications, measuring the amount of traffic sent by each flow in the network is essential. A network flow consists of all network packets that have the same flow identifier (ID). The flow ID can be extracted from the packet header and consists, e.g., of the source and destination IP addresses and ports. The size can be measured in terms of the number of packets in a flow or the byte volume. In this paper, we define flow size as the number of packets and flow bytes as the byte volume. Network flow measurement counts the total size of all packets in each flow. Examples of applications that use network flow measurement are traffic engineering, Distributed Denial of Service (DDoS) prevention, frequent items detection and heavy hitter detection.

Network flow measurement is turning out to be a research challenge in the advent of high-speed networks and devices. The main challenges are to deal with the high throughput requirements of Terabit Ethernet networks, which are defined as networks with speeds above 100 Gbps (Gigabits per second), and to minimize the memory resources of the counters that store the measurement results. Even though Dynamic RAM (DRAM) can fulfill the memory requirements [1], [2], moving from faster memories like Static RAM (SRAM) to slower off-chip DRAM cannot scale up to the bandwidth requirements of Terabit Ethernet networks. In fact, in high-speed networks, the high throughput and memory requirements make it impossible to keep the exact traffic count of each network flow.

Turning away from the conventional exact counting approaches, various probabilistic measurement approaches are available such as sampling [3]–[5], and sketches [6], [7]. Sampling-based measurement approaches [3]–[5] are commonly used and are helpful in reducing the memory requirements, but at the cost of lower accuracy and processing speed, especially when it comes to byte counting [8], [9]. Sketches are recently the most popular architectures for flow measurement, showing the best trade-off between accuracy, speed, and memory utilization. Many of the recently proposed sketch implementations [9]–[13] have shown success in flow measurement, but either at high computational/memory cost or at low throughput, causing a mismatch in throughput between the line speed and the measurement module. The high processing overhead of sketches is caused by multiple factors, including independent per-packet hash calculations, multiple memory accesses and arithmetic operations.

To reach the throughput requirements of Terabit Ethernet networks, we replace exact counters by approximate counters in the sketch, and we use FPGAs as the implementation platform. Recent advances in FPGA technology offer large memory bandwidth and logic density, which make FPGAs suitable platforms for high-speed network processing [14]. However, implementing sketches on FPGAs in a Terabit Ethernet network is challenging. An example of an authenticated encryption unit [15] shows that a network packet is processed per clock cycle at a frequency of 200 MHz to achieve a bandwidth of 200 Gbps on a Netcope FPGA platform [16].

We focus on the following challenges: (1) minimize the memory and arithmetic overhead of the sketch architecture to fit all storage and computation constraints in a single FPGA, and (2) maximize the throughput to operate in Terabit Ethernet networks. Therefore, we introduce the following contributions:

- A modified and hardware-friendly version of the Simple Active Counter (SAC) algorithm [17], which we call Hardware-oriented Simple Active Counter (HSAC), and a counter array based on HSAC.
- A novel sketch architecture, termed Approximate-CM sketch or A-CM sketch, inspired by the Count-Min sketch (or CM sketch) algorithm [7]. A-CM sketch uses approximate counters based on the HSAC algorithm and is optimized for high throughput by applying efficient memory organization and utilizing hardware-friendly hash computations, resulting in a high-speed flow measurement architecture on FPGA.
- A demonstrator on an FPGA in which the sketch hardware architecture has a direct Ethernet connection.

II. Background and Related work

A. Approximate Counters

Counters are an indispensable part of any statistical measurement. Approximate or probabilistic counting [18], [19] is one of the solutions to reduce the counting overhead at the expense of a small error in accuracy. This error is negligible when considering large amounts of data. Robert Morris, in 1978 [18], proposed a probabilistic counting technique to reduce the memory requirements of counters. The basic idea behind this algorithm is that the counter only stores an "order of magnitude approximation" of the actual count. Following this approach, many improved probabilistic counting techniques [5], [8], [17], [20]–[23] have been put forward. The main focus of most research was the improvement in accuracy. Algorithms such as CASE [21], DISCO [8], ICE Buckets [20], additive error counters [23] and SAC [17] show very high accuracy. However, these algorithms are not suitable for efficient hardware implementation. In our work, we present HSAC, a hardware-friendly version of the SAC algorithm.

B. Sketches

Sketches [6], [7], [11]–[13] have been shown to be a powerful solution for counting the number of times different events occur, with a good trade-off between computational resources and accuracy. Although, when the frequency of events becomes too large, the memory capacity of sketches gets affected. One way to eliminate this problem is by reducing the memory overhead by replacing the exact counters in the sketch with approximate counters, such as those discussed in Sect. II-A. Yang et al. [22] present a sketch design using approximate counter arrays which shows significant reduction in memory compared to sketches using exact counter arrays. However, the self-adapting counter approach by Yang et al. is a slower and more complex version of the SAC [17] algorithm, which is not suitable for hardware implementations in high-speed networks. In our work, we propose A-CM sketch, which is based on CM sketch [7] in combination with approximate counters. We perform further throughput optimization of the sketch architecture by manually exploring memory lay-out options, resulting in a minimal routing delay.

C. Network flow measurement on FPGA

Multiple performance metrics are important in the design of a hardware architecture. For network flow measurements, we consider three metrics: utilization of memory and logical resources, throughput, and accuracy. A few papers on hardware-based flow measurement architectures are available. Cache-based hybrid SRAM-DRAM architectures, proposed by Zadnik et al. [24], are suitable for passive flow measurements, where querying is done offline, and real-time flow measurement at line rate is not possible. Other related work on hardware concentrates specifically on sketch acceleration [25]–[28]. The throughput of the work of Lai [26] and Wellem [27] is bound to 10 Gbps on the NetFPGA platform. Saavedra et al. [28] propose the use of on-chip block RAMs (BRAM) along with DDR3 RAM and efficient pipelined hash computations on an FPGA. However, the use of external DDR3 RAM is not compliant with the throughput requirements of high-speed networks. Tong et al. [25] propose CM sketch and K-ary sketch based implementations for heavy hitter and heavy change detection, which achieves a throughput over 150 Gbps. However, the high memory utilization may prevent the design from increasing the width of the sketch over a certain limit. Similarly, Scotch [29] also achieves a higher throughput on FPGA, but at the cost of higher resource and memory consumption. A simple CM sketch implementation having a counter size of 32-bits would deplete the BRAMs in Virtex Ultrascale+ FPGAs when the width of the sketch exceeds $\approx$320,000. Prioritizing throughput alone at the cost of high memory consumption could be an impractical approach when it comes to resource-constrained devices like FPGAs. Our work is the first to achieve a throughput of 200 Gbps for 96-bit flow IDs (96-bit flow ID allows to distinguish individual flows, e.g. characterized by source and destination IP and source and destination port), while the aforementioned architectures are restricted to 32-bit flow IDs and do not reach 200 Gbps.

III. Algorithms

Our novel sketch architecture, Approximate CM sketch or A-CM sketch, is based on the Count-Min (CM) sketch data structure [7] and on the Hardware-oriented Simple Active Counter (HSAC) approximate counting algorithm. We explain both the existing CM sketch and our new HSAC algorithm.

A. Count-Min Sketch

Sketches are data structures consisting of multiple counter arrays. Count-Min (CM) sketch [7] is suitable for FPGA-based flow measurement architectures because of its relatively high throughput and small memory usage of $O(logN)$, where N is the number of different flows. A CM sketch is represented by a 2-dimensional array of counters with width w and depth d. The size of each counter is q bits.

When a network packet arrives, with size c_i and identified by flow ID $flow_i$, a total of d independent hash functions are computed on $flow_i$. Each hash value $h_j,\ j = \{1, ..., d\}$ is used as an index to determine which counter should be updated in the corresponding counter array row j. Each of these counters is then updated as: $counter[j, h_j] \leftarrow counter[j, h_j] + c_i$. When querying the sketch, the estimated count can be determined as $min(counter[j, h_j])$. CM sketch may overestimate the count, but the fact that underestimation will not happen is an advantage for large flow detection, which aims at detecting when a flow occupies more than its allowed bandwidth.

As explained in [7], the width and depth are determined by two parameters, namely the error factor (ϵ) and the error probability (δ), which are defined as follows: the additive error in answering a query is $\epsilon \times a_i$ with a bound probability $1 - \delta$, where a_i is the actual count value. Given the parameters ϵ and δ, the dimensions of w and d can be calculated as $w = \lceil \frac{e}{\epsilon} \rceil$ (with e representing Euler's number) and $d = \lceil \ln \frac{1}{\delta} \rceil$. The accuracy of the algorithm is thus determined by ϵ and δ.

B. Hardware-oriented Simple Active Counter (HSAC)

We introduce a hardware-friendly approximate counting algorithm - HSAC, which is a modified version of the Simple Active Counter (SAC) algorithm [17]. SAC divides the counter memory of q bits into exponent and estimation parts, namely exp and $count$ of sizes l bits and k bits respectively. The counter is updated with a random probability v and counter is incremented when $v < (\frac{inc}{2^{scale \times exp}} - \lfloor \frac{inc}{2^{scale \times exp}} \rfloor)$, where inc is the flow size. Re-normalization with a global scaling parameter $scale$ is applied when the counter overflows. The estimated value can be computed as $count \times 2^{scale \times exp}$. Although SAC can record large measurements with limited memory consumption, the presence of floating point divisions and multiplications, and the use of a single scaling factor for all the counters in an array make it less favourable for a high-throughput hardware implementation.

In HSAC, a fixed number of bits q is allocated in memory for each counter, similar to SAC. The difference with SAC is that the scaling parameter $scale$ is also included in q and is specific for each counter. As such, q is partitioned into an exponent part exp, an estimation part $count$, and a scaling part $scale$ of sizes l bits, k bits, and m bits, respectively. This way, HSAC avoids the unnecessary re-normalization of all counters when a single counter overflows, resulting in a better accuracy and lower computational overhead, in exchange for a slightly higher memory usage. Nevertheless, we show, in the remainder of this paper, that this slightly higher memory usage does not exceed the on-chip memory in the FPGA that we target.

The SAC algorithm possesses certain flaws for smaller values of l, i.e. the number of bits in the exponent part exp. For example, consider $l = 2$ and let the current exponent value be $exp = 3$. Also assume that $scale = 4$. At the next increment, the exponent becomes $exp = 2^l$, so re-normalization is required and the new value of exp is changed to $\lceil exp \times (\frac{scale}{scale+1}) \rceil$, which in fact is again 4 and the algorithm collapses. HSAC eliminates these flaws. Unlike in SAC, the values of $scale$ in HSAC are powers of 2. This eliminates the complexity of the re-normalization process in the algorithm. The multiplication and division operations are then replaced with shift operations which come for free in hardware. The pseudocode of HSAC for a single counter is shown in Algorithm 1. The number of bits required to store $scale$ is also reduced in HSAC by modifying the equation for estimation as $V = count \times 2^{(2^{scale} \times exp)}$.

HSAC supports both flow size and flow byte counting. For flow byte counting, it is necessary to keep the size of $count$ greater than the incoming packet size. Considering standard frame sizes of 1500 bytes, the size of $count$ is kept as 11 bits, whereas no such size restrictions apply for flow size counting. Anyway, it is better to keep the size of $count$ relatively large to avoid loss in accuracy. In SAC, re-normalization is required for all counters each time the value of $scale$ is updated. This could suspend the update process and cause loss of packets. In HSAC, in order to eliminate this issue and to reduce the processing overhead, $scale$ is unique for each counter. The use of separate scaling factors is similar to ICE Buckets [30], but in HSAC, each counter is considered a bucket, while ICE Buckets uses buckets of multiple counters with the same scaling factor for the whole bucket. Keeping $scale$ unique for each counter helps in maintaining the accuracy, and keeps the relative error in the measurement to a minimum. In the case where $scale$ is common for all the counters in an array as in SAC, a single flow with a large volume causes a large relative error to all other counters consisting of small flows.

Algorithm 1: Pseudocode of HSAC for a single counter

```
Parameters: size of the pseudo-random number - $S_p$, size of
  $scale$ (bits) - $m$, size of $exp$ (bits) - $k$, size of $count$ (bits) - $l$
Initialize(): $scale = 0$, $exp = 0$, $count = 0$
UpdateCounter($inc$):
  $F_1 = exp \ll (1 \ll (scale - 1))$; $count = count + inc \gg F_1$
  $p = (inc \,\&\, ((1 \ll F_1) - 1)) \ll (S_p - F_1)$; $v = rand()$
  if $v > p$
    $count = count + 1$
  if $count > (1 \ll k) - 1$
    $exp = exp + 1$; $F_2 = (1 \ll (1 \ll scale)) - 1$
    $p = count \,\&\, F_2 \ll (S_p - scale)$; $v = rand()$
    $count = count \gg (1 \ll scale)$
    if $v > p$
      $count = count + 1$
  if $exp == 1 \ll l$
    $exp = exp \gg 1$; $scale = scale + 1$ /*renomarlise counter*/
```

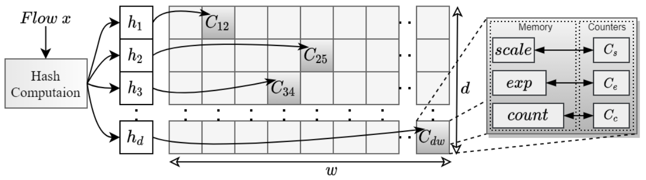

Fig. 1. Data structure of the Approximate-CM sketch (A-CM sketch)

IV. Hardware implementation

A high-level view of the A-CM sketch data structure that we propose is shown in Fig. 1. The counter array is a 2-dimensional array of size $w \times d$. Each element in the array is a counter of fixed size q. When a network packet enters the architecture, the extracted flow ID is hashed using the Xoodoo-NC algorithm [31], which has a low logical depth and calculates a hash output in one clock cycle. The output of Xoodoo-NC is split into d hash values of size $log_2 w$ to index the counters. The indexed counters are then updated. Instead of storing the exact value of each counter, HSAC is used.

To quantify our efforts, an implementation is made on a Xilinx Virtex Ultrascale+ FPGA (XCVU7P-FLVC2104-1-E). The memory array for the counters uses only on-chip dual port block RAM (BRAM), which is sufficient to satisfy the memory requirements of our architecture. Nevertheless, the number of BRAM blocks we need is large. This reflects in an increased placement and routing complexity when the memory and logic resources are spread over multiple clock regions, which, without proper consideration, leads to a decrease in the operating frequency. We employ efficient design approaches to mitigate this issue as described in the following paragraphs.

To tackle the routing delays, the counter array is organized in smaller memory units. Since the BRAM slots in FPGAs are fixed, building up larger memories in an automated way with

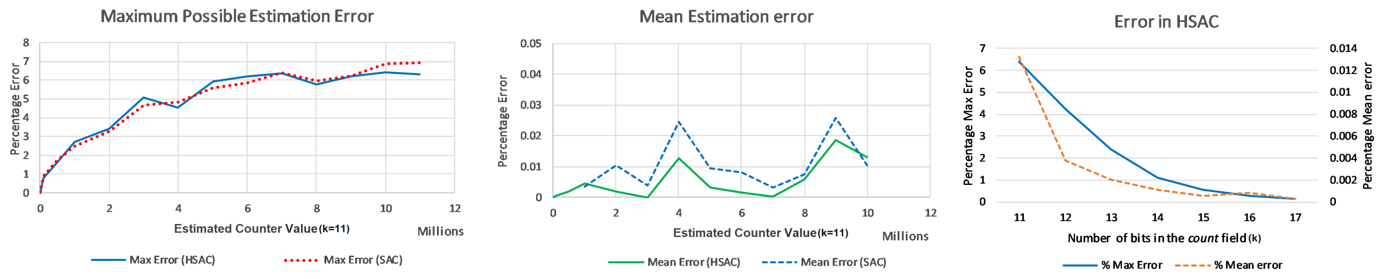

Fig. 2. (a) Maximum possible estimation error for HSAC vs. SAC; (b) Mean estimation error for HSAC vs. SAC (c) Maximum and mean estimation error for HSAC with a varying number of $count$ bits (k)

the design tools, minimizes the number of BRAM blocks used, but significantly increases the routing delay. Hence, instead of using one single large memory block for representing a counter, we employ separate memory units for each $scale$, exp, and $count$ value. During operation, these memories are read first before updating. All memory accesses are performed in parallel, which limits the total memory access delay to only two cycles for every update, one for the memory read and one for the memory write operation. Every $scale$, exp, and $count$ calculation has its own dedicated physical counter C_s, C_e, and C_c, respectively, i.e., the total number of counters used in the whole implementation is equal to $d.C_s + d.C_e + d.C_c$. These counters are reused in every update cycle.

Update process: When a flow x with a volume c arrives, one of the counters C_{dw} in each of the d arrays are updated by c. Which counters are updated is determined by the hash value h_j, where $1 \leq\ j \leq\ d$. The memory location of the counter value to be updated is indexed by the hash value. After adding the packet volume c, the updated value is written back to the memory. All the memory read and write operations are performed in parallel so that the latency is only one clock cycle for each read and write operation. To minimize the routing delay, a buffer stage is added in between the memory read and addition stage, which takes one clock cycle for registering the intermediate values. The latency in terms of number of clock cycles for each update operation is five. All the stages are pipelined, so that the sketch unit can accept and process a 96-bit flow ID in every clock cycle at line rate.

Query process: Each counter value consists of a fixed number of bits, i.e., the concatenation of $scale$, exp, and $count$. The most significant m =1 or 2 bits are used for $scale$, the next l bits are used for exp, and the least significant k bits are used for $count$. Similar to the update process, the hash computation and memory read operations are performed and the values of $scale$, exp, and $count$ are read from the memory. These values are concatenated to obtain the count values for each of the d counters C_{dw}. The minimum of d count values is computed and taken as the output. The estimated value is computed from the minimum of all the d values. The querying process has a latency of 3 clock cycles.

HSAC implementation: The core component of the sketch is the approximate counter HSAC. The HSAC block consists of an update, a re-normalization, and a Linear Feedback Shift Register (LFSR) module. The update and re-normalization processes are discussed in Sect. III-B. A S_p-bit LFSR with XOR feedback is used as the pseudo-random number generator. The output size of the LFSR is equal to $2^m.(2^l - 1)$ bits ($S_p = 2^m.(2^l - 1)$) to avoid any loss in accuracy, where m and l are the number of bits for $scale$ and exp, respectively.

V. Evaluation and Results

We present the results of three experiments. The first experiment analyzes the estimation error and the performance of HSAC compared to SAC. In the second experiment, the functionality of the A-CM sketch is tested on a lab setup with network traffic applied to an FPGA board that hosts the sketch. In the third experiment, the performance and resource utilization of the architecture are evaluated using FPGA design tools. We chose the error factor and probability similar to related work, in order to make a fair comparison. The parameters d and w are chosen accordingly. Each HSAC counter has a width of 16 bits, where $l = 4$, $k = 11$, and $m = 1$, and can count up to 2.197x10^{12}.

A. Analysis of HSAC

We examine the error rate of HSAC in comparison with SAC, resulting in the maximum possible estimation error and mean estimation error for a single counter shown in Fig. 2(a,b). For a counter array, the error will be higher for SAC because the re-normalization will be applied to all the counters irrespective of the counter values, whereas the error remains constant for HSAC as only the overflown counter will be re-normalized. For a 16-bit counter (l=4, k=11, m=1) with a latency of one clock cycle, HSAC occupies 139 LUTs and 38 FFs at 625 MHz, compared to 171 LUTs and 46 FFs at 303 MHz for SAC. The maximum estimation error is reduced remarkably with increasing $count$ bits (k) as shown in Fig. 2(c). The measurement error of HSAC is quite small and thus does not introduce the danger of false negatives for A-CM sketch. For a count of 1×10^7, when k increases from 11 to 14 bits, the worst case estimation error reduces from 6.4% to <1% with the mean error being very close to zero.

B. Functional test of A-CM sketch

The functionality of the proposed A-CM sketch implementation is tested on a lab setup. This setup consists of a Xilinx VC707 board (with a Virtex-7 FPGA), which is attached to a Gigabit network switch. Traffic to and from the FPGA is generated through a Python program. In the System-on-Chip,

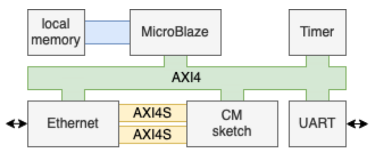

Fig. 3. System on chip for functional test

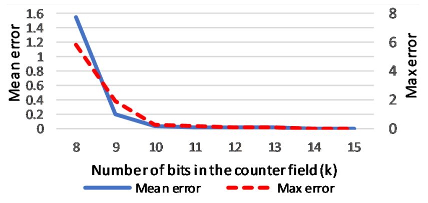

Fig. 4. Accuracy variation of A-CM sketch compared to CM sketch

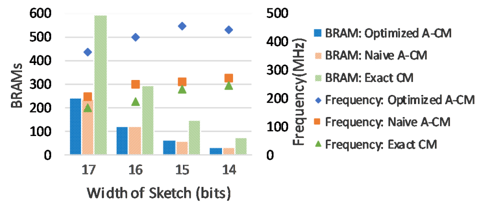

Fig. 5. BRAM utilization and frequency as a function of the sketch width

shown in Fig. 3, the CM sketch has direct connections for receiving and sending frames in the network. It is connected through AXI4S with the Ethernet core. After filtering out UDP and TCP frames, network packets are processed in the sketch at line rate. In order to test the functionality, a MicroBlaze softcore processor queries a specific flow by sending the flow ID to the sketch, which then returns the estimated count. This communication happens over an AXI4 bus, which is also connected to a Timer and a UART. The UART is used for debugging. The functionality test is not performed at maximum throughput, but on a 1 Gbps lab setup.

C. Implementation results of A-CM sketch

In order to evaluate the performance of the sketch, we use Vivado 2017.4 to generate implementation results for a Xilinx Virtex Ultrascale+ FPGA (XCVU7P-FLVB2104-2-i).

Resource utilization and performance: By replacing exact counters with approximate counters, the memory footprint of the CM sketch significantly decreases. For A-CM sketch with $d = 4$ and $w = 2^{16}$, using 16-bit approximate counters, the total memory requirement is 512 KB, whereas with exact counting, 41 bits are required to represent the same counter value and the memory requirement is 1312 KB (on a similar hardware architecture of the sketch, with the only change being the counters). Also the frequency of operation is significantly improved, as shown in Table I. The exact counter based implementation utilizes 2.47× more BRAM while having a latency that is 2.2× higher than the approximate counter based implementation. The difference in accuracy is very much negligible when compared to the corresponding CM sketch implementation. The mean and maximum accuracy variation of A-CM sketch versus a 24-bit CM sketch implementation almost touches zero when the number of counter field bits k is greater than or equal to 10, as shown in Fig. 4.

To analyze the efficacy of the memory optimization techniques introduced in Sect. IV, the A-CM sketch is compared with a naive A-CM sketch implementation (no memory optimization) and an exact counter based CM sketch. The results in Fig. 5, show the improvement in operating frequency of the optimized A-CM sketch compared to the naive A-CM sketch. Table I shows that the naive A-CM sketch consumes less logic resources.

Throughput: The optimized A-CM sketch implementation having w=2^{16} achieves a maximum theoretical throughput of ≈200 Million packets per second (Mpps) on a single port of the NFB-200G2QL [16] platform, and a total of ≈400 Mpps with each port having its own instance, assuming the worst case scenario where one packet is received in every cycle. It can process a 96-bit flow ID per clock cycle irrespective of the packet size. Assuming a minimal packet size of 64 bytes, a single instance of A-CM sketch can deliver a minimum throughput of ≈233 Gbps (454Mpps) and ≈212 Gbps (414Mpps) respectively for w=2^{15} and w=2^{16}, and d=4, with the only bottleneck in limiting the throughput being the network interface. Table II shows that our implementation outperforms state-of-the-art sketches on FPGA.

TABLE I
COMPARISON OF A-CM SKETCH WITH EXACT (FULL-PRECISION) COUNTER BASED CM SKETCH

	w (d=4)	LUT	FF	BRAM	Operating Frequency	Update Latency
CM sketch	65536	1841	438	296	188.6 MHz	26.50 ns
	32768	1301	414	148	232.5 MHz	21.50 ns
A-CM sketch Optimized	65536	1830	784	120	414.9 MHz	12.05 ns
	32768	1512	740	60	454.5 MHz	11.00 ns
A-CM sketch Naive	65536	1408	391	120	250.0 MHz	20.00 ns
	32768	1211	375	60	260.4 MHz	19.20 ns

TABLE II
THROUGHPUT COMPARISON OF FPGA BASED SKETCH IMPLEMENTATIONS

Platform	Sketch	Flow-ID Size	d, w	Throughput(Gbps)	ϵ	$1-\delta$
Virtex-5 [27]	K-ary	32-bit	$1, 2^{16}$	53	-	-
Virtex Ultrascale [25]	K-ary	32-bit	$5, 2^{16}$	159	-	-
Virtex Ultrascale [28]	CM	32-bit	$4, 2^{14}$	128	0.0001	0.98
Virtex Ultrascale [25]	CM	32-bit	$5, 2^{16}$	155	0.00004	0.99
Ours,Virtex Ultrascale+	CM	96-bit	$4, 2^{15}$	233	0.00008	0.98
Ours,Virtex Ultrascale+	CM	96-bit	$4, 2^{16}$	212	0.00004	0.98
Ours,Virtex Ultrascale+	CM	96-bit	$5, 2^{16}$	196	0.00004	0.99

VI. CONCLUSION

In this paper, we present A-CM sketch (Approximate Count-Min sketch), an ultra-high-speed FPGA implementation of the CM sketch algorithm using approximate counters. Thanks to our architectural optimization efforts in the CM sketch data structure and our algorithmic optimization efforts, resulting in the HSAC (Hardware-oriented Simple Active Counter) algorithm, we manage to obtain an FPGA implementation that outperforms, to our knowledge, all existing sketch-based data structures on FPGA. Moreover, our implementation is compliant to a 200 Gbps network interface, making it suitable for network flow measurement at line rate in Terabit Ethernet networks.

VII. ACKNOWLEDGEMENT

This work is supported by the ESCALATE project, funded by FWO and SNSF (G0E0719N), and by CyberSecurity Initiative Flanders (VR20192203).

References

[1] Sriram Ramabhadran and George Varghese. Efficient implementation of a statistics counter architecture. In *in Proc. ACM SIGMETRICS*, 2003.

[2] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *Proc. of ACM SIGMETRICS '06, France*, 2006.

[3] CISCO. CISCO IOS NetFlow Version 9. http://www.cisco.com/c/en/us/products/ios-nx-os-software/netflow-version-9/index.html, 2015.

[4] sFlow. Traffic Monitoring using sFlow. http://www.sflow.org/sFlowOverview.pdf, 2003.

[5] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *IEEE INFOCOM*, pages 26–30, 2008.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer Berlin Heidelberg, 2002.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[8] 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 *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 665–674, 2010.

[9] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with Univ-Mon. *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, pages 101–114, 2016.

[10] Q. Huang et al. SketchVisor: Robust network measurement for software packet processing. *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, pages 113–126, 2017.

[11] T. Yang et al. Elastic sketch: Adaptive and fast network-wide measurements. *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, pages 561–575, 2018.

[12] Z. Liu et al. Nitrosketch: Robust and general sketch-based monitoring in software switches. *Proc. ACM SIGCOMM*, pages 334–350, 2019.

[13] Simon Scherrer, Che-Yu Wu, Yu-Hsi Chiang, Benjamin Rothenberger, Daniele E Asoni, Arish Sateesan, Jo Vliegen, Nele Mentens, Hsu-Chun Hsiao, and Adrian Perrig. Low-rate overuse flow tracer (loft): An efficient and scalable algorithm for detecting overuse flows. In *Proceedings of the 40th International Symposium on Reliable Distributed Systems (SRDS) (to appear)*, 2021.

[14] Y. Li, R. Miao, C. Kim, and M. Yu. FlowRadar: A Better NetFlow for Data Centers. In *13th USENIX Symposium on Networked Systems Design and Implementation*, pages 311–324, 2016.

[24] M. Zadnik, M. Canini, A. W. Moore, D. J. Miller, and W. Li. Tracking elephant flows in internet backbone traffic with an FPGA-based cache. In *FPL*, pages 640–644, 2009.

[15] Z. Martinasek, J. Hajny, D. Smekal, L. Malina, D. Matousek, M. Kekely, and N. Mentens. 200 Gbps hardware accelerated encryption system for FPGA network cards. In *ASHES*, page 11–17. ACM, 2018.

[16] Netcope. NFB-200G2QL FPGA-based Hardware . www.netcope.com/getattachment/bb2b8efa-9925-438d-b895-897d7c1e4745/NFB-200G2QL-product-brief.aspx.

[17] R. Stanojevic. Small active counters. In *IEEE INFOCOM*, pages 2153–2161, 2007.

[18] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, pages 840–842, 1978.

[19] P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[20] G. Einziger, B. Fellman, and Y. Kassner. Independent counter estimation buckets. In *IEEE INFOCOM*, pages 2560–2568, 2015.

[21] Y. Li, H. Wu, T. Pan, H. Dai, J. Lu, and B. Liu. CASE: Cache-assisted stretchable estimator for high speed per-flow measurement. In *IEEE INFOCOM*, pages 1–9, 2016.

[22] T. Yang, J. Xu, X. Liu, P. Liu, L. Wang, J. Bi, and X. Li. A generic technique for sketches to adapt to different counting ranges. In *IEEE INFOCOM*, pages 2017–2025, 2019.

[23] R.B. Basat, G. Einziger, M. Mitzenmacher, and S. Vargaftik. Faster and more accurate measurement through additive-error counters. In *arXiv:2004.10332 [cs.DS]*, 2020.

[25] D. Tong and V. K. Prasanna. Sketch acceleration on fpga and its applications in network anomaly detection. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):929–942, 2018.

[26] Y.-K. Lai, N.-C. Wang, T.-Y. Chou, C.-C. Lee, T. Wellem, and H. T. Nugroho. Implementing on-line sketch-based change detection on a NetFPGA platform. In *1st Asia NetFPGA Developers Workshop*, 2010.

[27] T. Wellem, Y. Lai, C. Huang, and W. Chung. A hardware-accelerated infrastructure for flexible sketch-based network traffic monitoring. In *IEEE HPSR*, pages 162–167, 2016.

[28] A. Saavedra, C. Hernández, and M. Figueroa. Heavy-hitter detection using a hardware sketch with the Countmin-CU algorithm. In *Euromicro DSD*, pages 38–45, 2018.

[29] Martin Kiefer, Ilias Poulakis, Sebastian Breß, and Volker Markl. Scotch: Generating fpga-accelerators for sketching at line rate. *Proceedings of the VLDB Endowment*, 14(3):281–293, 2020.

[30] G. Einziger, B. Fellman, and Y. Kassner. Independent counter estimation buckets. In *IEEE INFOCOM*, pages 2560–2568, 2015.

[31] Arish Sateesan, Jo Vliegen, Joan Daemen, and Nele Mentens. Novel bloom filter algorithms and architectures for ultra-high-speed network security applications. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 262–269. IEEE, 2020.