Implementing On-line Sketch-Based Change Detection on a NetFPGA Platform

Yu-Kuen Lai, Nan-Cheng Wang, Tze-Yu Chou
Chun-Chieh Lee, Theophilus Wellem*, Hargyo Tri Nugroho*
Dept. of Electrical Engineering, Dept. of Electronic Engineering*
Chung-Yuan Christian University
Chung-Li 32023
Taiwan
{ylai, g9778048, g9778042, g9818019, g9776070, g9776071}@cycu.edu.tw

ABSTRACT

Sketch-based algorithms are widely applied in various networking applications. In this paper, we present a compact implementation for on-line traffic change detection on a NetFPGA platform. The system utilizes a straight forward scheme to reveal the key of flow with tradeoff on the accuracy for hardware simplicity. It is capable of digesting traffic up to 4Gbps line rate with accuracy needed based on the available memory on-board.

Categories and Subject Descriptors

C.2.1 [**Computer-Communications Network**]: Network Architecture and Design

Keywords

Data Stream, Sketch, Change Detection, NetFPGA

1. INTRODUCTION

Monitoring and identifying abnormal behaviors in network traffic are important tasks for network security. As network bandwidth grows exponentially, the scaling of monitoring and measuring capabilities for collecting accurate statistics becomes a critical issue [17]. The challenge we are facing is to process a potentially unlimited amount of data in a limited time and space. In addition, each element or record of the data stream might have only one chance to be examined.

In this work, we focus on one of the interesting behaviors: monitoring the presence of abrupt changes in network traffic. Internet traffic can be naturally regarded as a data stream since packet data arrives rapidly as a series of elements. Abrupt changes in network traffic pattern might indicate anomalies or malicious activities, such as port scan or denial of service (DoS) attacks [11]. This phenomena is also known as *heavy changer.* It is a network flow whose change in traffic volume between two monitoring interval exceeds a predefined threshold [5]. A network *flow* can be defined as a stream of packets with some common attributes. For example, it can be packets having the same pair of source and destination IP address, or packets consist of the same 5-tuple attributes: the source and destination IP address, the source and destination port, and the protocol number.

Based on the sketch algorithm proposed by Krishnamurthy et al. [20], we demonstrate a system which is capable of detecting changes of flow volume in on-line fashion at wire speed on NetFPGA reference platform. The remainder of the paper is organized as follows. In Section 2, we first introduce some general backgrounds of sketch algorithm and universal hash function. The system architecture and implementation details are described in Section 3. Experiments are conducted with discussions presented in Section 4. We also provide the related work in Section 5. Finally, we conclude the paper with future work in Section 6.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
1st Asia NetFPGA Developers Workshop 13-15 June 2010, KAIST, Daejeon, South Korea
Copyright 2010 ACM X-XXXXX-XX-X/XX/XX ...$10.00.

2. BACKGROUND

A data stream $\phi = (a_1, a_2, ...)$ is a massive sequence of elements. Each element, $a_t = (k_t, u_t)$ consists of a *key,* k_t and an *update,* u_t. The key k_t in the data stream model can be used to represent the traffic flow. Typically, sampling [15, 16] is a popular method applied to tackle the processing and storage cost incurred by the huge amount of traffic. Sketch is another interesting approach as it offers low space requirement and guaranteed accuracy.

2.1 Sketch Algorithm

Sketch [7, 2, 25] is a powerful yet compact data structure capable of synopsizing substantial numbers of data elements without keeping its stateful information. The sketch algorithms rely on the probabilistic property of universal class of hash functions [6] to guarantee its accurate estimation on various attributes of the data streams. Therefore, it is widely used in many high-speed network applications [11, 14, 22, 5, 4, 20].

Take the K-ary [21] sketch $S(t)$ for example, it utilizes a two-dimensional array of counters $C[i][j]$, where $0 \leq i < H, 0 \leq j < K$. The parameter H represents the number of arrays and K is the number of entries within each array. The two-dimensional array of counters is indexed by a set of 4-Universal hash functions $H = \{h_i, 0 \leq i < H\}$. Each hash function h_i maps a key $k \in \{0, 1, \cdots, n-1\}$ into the hashing space of $\{0, 1, \cdots, K-1\}$.

Initially, all the counters are set to zero. As each element $a_t = (k_t, u_t)$ of the data stream arrives, the key k_t is hashed by the set of H hash functions. Then, the value of u_t is added to the set of counters indexed by the H hash outcomes, as illustrated in Equation (1).

$$C[i][h_i(k_t)] = C[i][h_i(k_t)] + u_t, \forall i \in [H] \quad (1)$$

The sketch data structure $S(t)$ contains the final accumulated values in these counters within a specific observation interval $\triangle T$. A query can be conducted to estimate a specific attribute on the data stream based on the sketch collected. We refer to the original work[21] for more details.

Due to its linearity, we can also combine several sketches together for query processing. A detailed analysis and comparison of various sketch algorithms can be found in Cormode's survey [9, 8].

2.2 Universal Hash Function

Carter and Wegman first described the idea of a universal class of hash functions [6] in 1979. The universal class of hash functions is a family of hash functions with a special randomized property. Given two keys, the probability of hashing these two keys into a same value is bounded as long as the function is randomly selected from the family. The family of 2-universal hash functions has the collision probability specified in a pairwise independent manner. That is, for all $x \neq y \in U$ and $s, t \in B$, $P_{h \in H}[h(x) = s,\ h(y) = t] = \frac{1}{B^2}$.

The universal class of hash functions can be generalized as k-universal defined in Equation 2, where $x < p$ and p is a prime number. The parameter a_i is selected randomly where $0 < a_i < p$.

$$h(x) = \sum_{i=0}^{k-1} ((a_i x^i) \bmod p) \bmod m \quad (2)$$

$$P_{h \in \mathcal{H}}[h(x_i) = v_i] = \frac{1}{m^k} \quad , where\, v_i \in [m],\ i \in (0, 1, \cdots, k-1)$$

In general, the prime number, p is chosen to be a Mersenne prime such as $2^{31} - 1$, $2^{61} - 1$ or $2^{89} - 1$ to avoid the long latency of division computation.

The tabulation method is another popular way suitable for hardware implementation [29]. For example, in order to hash a n-byte string of $x_0 x_1 \cdots x_{n-1}$, we need a $256 \times n$ 2-D array a_t. Each single column consists of 256 pre-calculated hash values by using a hash function randomly drawn from a *2*-universal hash family. The whole table is indexed by each byte value of x_i and position of i in the string. The hash process, shown in Equation 3, is done by XORing a sequence of values $a_t[x_i][i]$, where $i \in (0, 1, \cdots, n-1)$.

$$h_{tab}(x_0 x_1 \cdots x_{n-1}) = a[x_0][0] \oplus a_t[x_1][1] \oplus \cdots \oplus a_t[x_{n-1}][n-1] \quad (3)$$

Thorup and Zhang [32] proposed a tabulation method for 4-universal hash function. For a given 32-bit key k, it is divided into two 16-bit subkeys, a and b. Three 4-universal hash functions h_0, h_1, and h_2 are precomputed and tabulated. The hash process can be proceeded by table lookup on the two subkeys and exclusive-or operation shown as follows: $h[k] = h_0[a] \oplus h_1[b] \oplus h_2[a+b]$.

3. IMPLEMENTATION

Based on the detection scheme proposed by Krishnamurty et al. [20], the system is designed to be a network probe with four observation points [1] for IP traffic. The architecture is partitioned into two major parts, denoted as NetFPGA hardware and system software.

The NetFPGA hardware consists of a sketch module which is placed behind the *Input Arbiter* in the NetFPGA reference pipeline. The data structure, known as the *Observed Sketch* $S_o(t)$, is stored in the NetFPGA on-board SRAM to record the update value during the observing time interval.

The system software, executed on host computer side, is in charged of performing statistical operations based on the Observed Sketch $S_o(t)$. A series of *Forecast Sketch* $S_f(t)$, *Forecast Error Sketch* $S_e(t)$ and threshold T_A are computed at the end of each observing time interval $\triangle T$. A typical observing time interval can be ranging from 60 seconds to 300 seconds.

The system achieves *on-line*, *one-pass* detection by utilizing the current incoming key to query the previous Forecast Error Sketch $S_e(t)$ on the NetFPGA hardware. If the query result is higher than the threshold T_A, a significant change for that given flow is found. The corresponding key is recorded and alarm raised.

3.1 Sketch Module

The sketch update datapath is the critical bottleneck for system operation since the process has to be done in timely manner for each new frame arrived. The user datapath (UDP) consists of 64-bit data bus operated in clock frequency of 125Mhz. Therefore, the system has to process the sketch update for every $T_{frame} = 21$ clock cycles. This is assuming a worst-case scenario of 4-port inbound network traffic of minimum-sized Ethernet frame with inter-frame gap and preamble of 96 and 64 bit time respectively. The detailed block diagram of Sketch module is shown in Figure 2. It consists of a FIFO buffer and blocks of hash function pipeline. Upon arrival of a frame, both of the source IP address and packet length are extracted from the header. They are kept in the FIFO buffer to hide the latency of hash calculation.

The system consists of three sets of standalone hash function ($H = 3$). The hashed values are used as indexes for updating the observed sketch data structure in SRAM. For each time interval $\triangle T$, an observed sketch $S_o(t)$, is created and recorded in a ping-pong fashion. As shown in the Figure 2, the system maintains two observed sketch data structures in the SRAM. While the current observed sketch $S_o(t_2)$ is being updated, the other sketch $S_o(t_1)$ of the previous time interval can be retrieved by the host via PCI bus. Besides, the hashed values are used to query the error sketch $S_e(t)$ produced from the host processor. The purpose is to detect the abrupt change in real-time and record the key in one-pass fashion. The details will be explained in the next section.

3.2 System Software

At the end of each time interval $\triangle T$, the observed sketch $S_o(t)$ are moved to the host through PCI bus. The host processor keeps all the sketch data structures to compute the forecast sketch $S_f(t)$ by using desired smoothing model with window size W. The processes are indicated at the steps (2) and (3) shown in Figure 1.

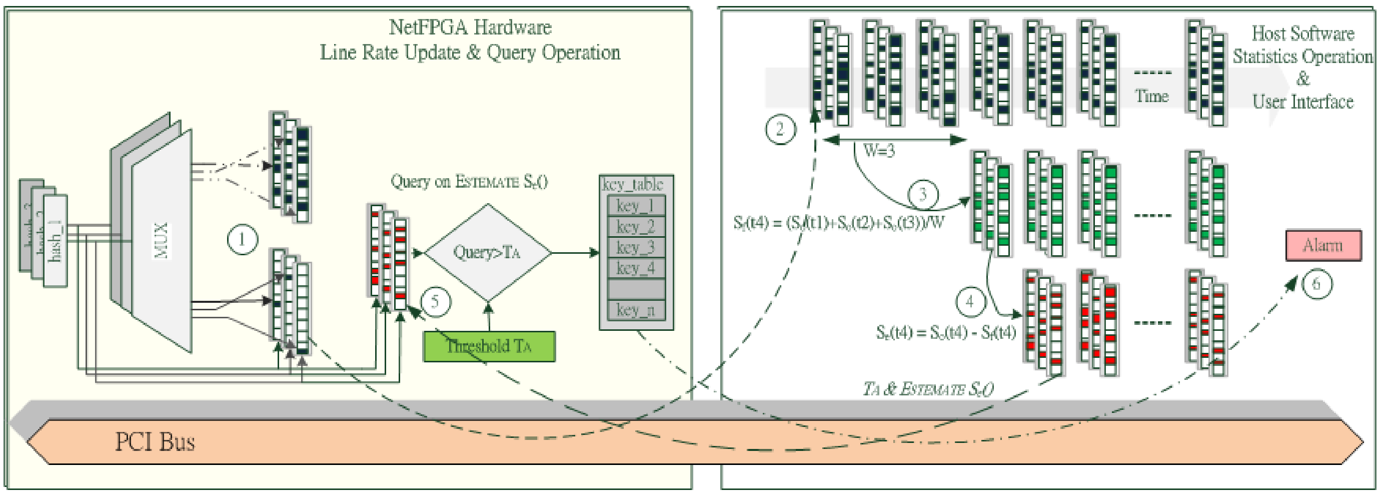


Figure 1: System operations

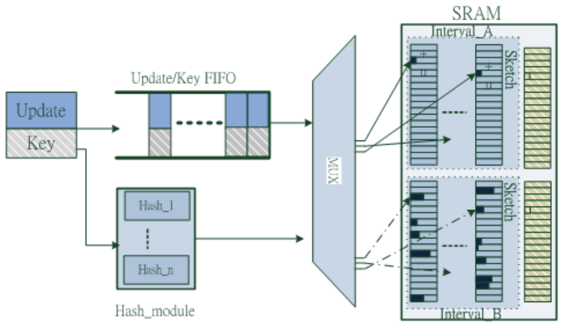


Figure 2: The datapath of Sketch Update module

$$S_f\left(t\right) = \frac{\sum_{i=1}^{W} S_f\left(t-i\right)}{W},\ W \geq 1 \tag{4}$$

We use the simple smoothing model: *Moving Average* (MA) to compute the Forecasting Sketch $S_f\left(t\right)$. The Error Sketch is constructed by subtracting the Forecasting Sketch $S_f\left(t\right)$ with the current Observing Sketch $S_o\left(t\right)$ shown in the equation as follows: $S_e\left(t\right) = S_f\left(t\right) - S_o\left(t\right)$. The host processor further calculates two sets of sketch array: (1)$ESTIMATE(S_e\left(t\right))$ and (2)$ESTIMATEF_2(S_e\left(t\right))$ shown in the Equation (5) and (7) respectively.

$$ESTIMATE(S_e\left(t\right)) = \left\{\frac{K}{K-1}C_{Se}[i][j] - \frac{1}{K-1}\sum_{j\in[k]} C_{Se}[0][j]\right\} \tag{5}$$

$$F_2^{hi}(S_e\left(t\right)) = \frac{K}{K-1}\sum_{j\in[k]}(C_{Se}[i][j])^2 - \frac{1}{K-1}\sum_{j\in[k]}(C_{Se}[0][j])^2 \tag{6}$$

$$F_2^{estimate} = median_{i\in[H]}\left\{F_2^{hi}\right\} \tag{7}$$

The alarm threshold for each observing time period, T_A is obtained based on the square root of the estimated second moment $F_2(S_e\left(t\right))$ times a predetermined parameter T.

$$T_A = T.\left[F_2^{estimate}(S_e\left(t\right))\right]^{\frac{1}{2}} \tag{8}$$

In order to perform fast on-line query on each incoming frame, the value of alarm threshold T_A is kept in the hardware register. The estimated Error Sketch $ESTIMATE\, S_e\left(t\right)$ is also stored in the SRAM.

For each key, the query is conducted against the estimated Error Sketch in hardware. The query result, representing the magnitude of change, is compared with the threshold T_A. If this value is above T_A, the alarm is raised with the key recorded in the key table. The table will be passed to the host processor at the end of time epoch. The processes are illustrated at step (5) and (6) shown in Figure 1.

4. EXPERIMENTS AND DISCUSSIONS

The quality of hash function plays an important role on the estimation accuracy. In order to compare the latency and resource utilization on NetFPGA platform, we implement two different types of hash function: the 2-Universal and 4-Universal. The implementation further includes the tabulation and pipelined multiplier schemes, illustrated in Equation (2) and (3).

Shown in Table 3, we can see there is a difference of about 20% on the slices and 4-Input LUTs utilization for 2-Universal and 4-Universal tabulation implementation. This is mainly due to three precomputed tables for 4-Universal hash lookup. The multiplier implementation utilizes the CW-trick [6] with embedded multipliers provided on the Virtex-II Pro FPGA in pipelined fashion. The hash pipeline is capable of processing a 32-bit key on each UDP clock cycle with total hashing latency approximately 4 times longer than that of the tabulation counterpart.

We first verify the NetFPGA hardware for line-rate sketch update by sending frames generated from a 4-Port 1000-BaseT Gigabit module hosted on the IXIA 1600 Traffic Generator chassis. We observed 0.16% packet lost under the stress test with 4Gbps minimum-sized frame.

Trace-driven experiments are also conducted based on the network traces from MAWI Working Group Traffic Archive [23]. The first 15-minute trace consists 51,778 flows of distinct source IP addresses. It's part of a 24-hour-long trace in a trans-Pacific line (18Mbps CAR on 100Mbps link) [24] collected at MAWI's samplepoint-B. Another 15-minute trace (200904022100.dump) file is adopted from a 150Mbps link at MAWI's samplepoint-F. It consists 286,369 flows of distinct source IP addresses.

The system under test is configured with three hash functions (H=3) and three sets of 4096-entry (K=4k) of sketch table. Each entry is a 32-bit counter. Thus, the total size of the sketch data structure is 48k bytes. Each of the sketch table can be scale up to 32k entries (K=32k) depend on the accuracy required.

The observing time period is sixty seconds ($\triangle T = 60$) with moving average window size of three ($W = 3$). Traces are replayed by tcpreplay-3.4.4 [33] through Intel Pro-1000MT dual-port server adapter. The results are verified with our simulator.

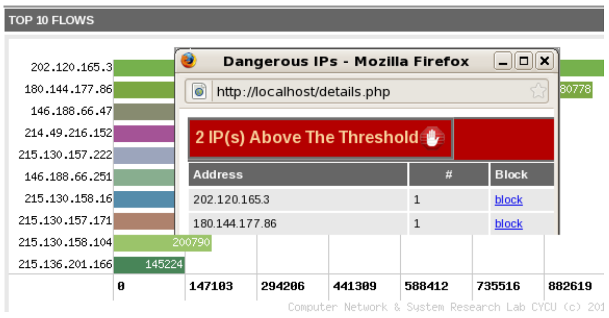
We build a simple web-based interface showing top 10 flows and displaying a list of IPs which is detected over the predefined threshold. This interface, shown in Figure 3, is built using PHP and Javascript. Pages are reloaded every observing interval automatically.

Despite the small sketch size and number of hash functions, the system can successfully identify several source IP addresses whose change is greater than the predefined threshold (T fraction of the L2 norm of the error sketch). We present some of the identified flows (at the bottom marked with IP and time interval) against the total traffic volume (top) in Figure 4.

One main issue in the sketch-based methods is that we have to identify the keys with significant change. Cormode and Muthukrishnan [12] proposed a scheme based on group testing to find items that have large difference (deltoids) in high-speed network traffic. The data structure can be viewed as an extension to k-ary sketch with multiple counters in each hash table. Based on the implementation [12], the memory requirement is between 500 Kb to 3 Mb per stream collection (group).

Another approach is the reversible sketch scheme proposed by Schweller et al. [30]. With parameters of (H=6, K=64k) and (H=6, K=4k), the scheme requires memory sizes of 3 MB and 192 KB, respectively, to support the IP mangling and modular hashing operation.

The way we uncover the original key of the flow is to query the error sketch of the previous observing interval and compare the threshold T_A on the arrival of each new frame. The system consumes only 48KB of memory in total for sketch data structure (H=3, K=4k). By comparing to the schemes proposed by reversible sketch [30] and combinatorial group testing [10], its simplicity is much attractive and manageable for hardware implementation under limited resources. Therefore, we are interested in the tradeoff where extra compromise in accuracy might occur. Figure 5 shows the simulation results on the false-positive and false-negative rate based on the scheme we adopt. The false negative error rate is slightly higher than the false positive rate. It's because the key of the flow, which is updated in the previous time epoch, does not show up again at the current observing interval.


TOP 10 FLOWS

Dangerous IPs - Mozilla Firefox

http://localhost/details.php

2 IP(s) Above The Threshold

Address	#	Block
202.120.165.3	1	block
180.144.177.86	1	block

Figure 3: The User Interface

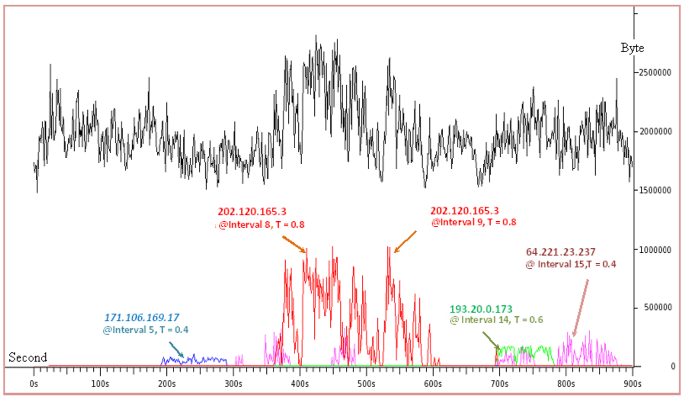


Figure 4: Selected source IP addresses above the threshold for various threshold T. The observing time period is 60 seconds with window size of three (W=3).

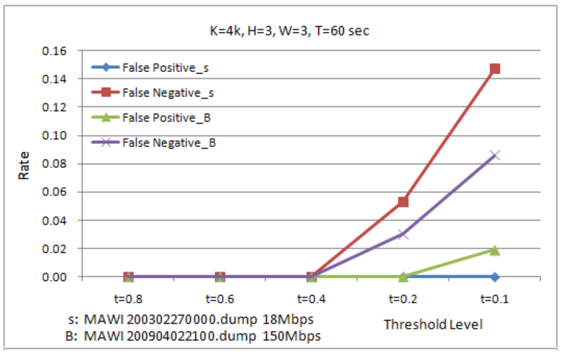


Figure 5: False positive and false negative error rate.

5. RELATED WORK

The original implementation provided by Krishnamurthy et al. [20] is a software-based design for off-line operation. With configuration of $H = 5$ and $K = 64K$, the running time for processing 10 million hash computations is 0.45 second on a 900Mhz Ultrasparc-III running Solaris 5.8 operating system. Schweller et al. [30, 31] later presented a change detection system based on a reversible k-ary sketch scheme. The objective of the reverse hashing is to infer the key of flows from the sketch. The system is implemented in hardware development board using three Xilinx Virtex

Table 1: FPGA utilization. (pipelined 4-Universal hash functions, H=3, K=32k)

Resources	Utilization	Percentage
Slice Registers	22,687 out of 47,232	48%
4 input LUTs	20,433 out of 47,232	43%
Occupied Slices	16,671 out of 23,616	70%
RAMB16s	144 out of 232	62%
MULT18X18s	72 out of 232	31%
IOBs	356 out of 692	51%

Table 2: System accuracy with various threshold parameter T. (pipelined 4-Universal hash functions, H=3, K=32k)

T	0.8	0.6	0.4	0.2	0.1	0.05	0.02
False Positive	0	0	0	0	0	0.005	0.008
False Negative	0	0	0	0.05	0.11	0.14	0.17

2000E FPGAs and hosted in a Solaris Ultra-10 workstation. With 2-Universal hash functions and key of 32-bit IP address, the hardware ($H = 5$, $K = 4k$) can achieve throughput of 16.1Gbps with modular hashing and IP mangling for 40-byte IP packets.

A substantial change detection literature [12, 19, 26] relies on the advantage of the sketch-based algorithm to summarize the high-dimensional nature of network data stream. Therefore, it's widely used in many network security related applications and intrusion detection systems to figure out many aspects of a network traffic's behavior in addition to the signature-based detection scheme [13]. Nguyen et al. proposed an FPGA-based feature extraction module for network flow monitoring [27]. The estimate procedure of the sketch is similar to that of Count-Min [9] sketch. The system was implemented based on Xilinx Virtex-II XC2V1000 and tested with various combination of H and K value. The system achieves throughput of 21.25 Gbps and accuracy of 97.61% with configuration of $H = 4$ and $K = 16k$. Pati et al. [28] proposed the same architecture as that in [27]. The system utilizing the sketch based on Bob Jenkins' 32-bit hash function, was implemented on a Xilinx ML310 board which contains VirtexII-Pro XC2VP30 FPGA. Their experiment run in off-line mode with throughput of 3 Gbps using the same range of H and K in [27]. Barman et al. proposed sketch-based techniques to detect attacks in routers [3]. The techniques are based on change detection and leverage the accurate estimation provided by Count-Min[9] and FM [18] sketches. No hardware implementation is presented in the paper.

6. CONCLUSION AND FUTURE WORK

This paper presents an implementation of on-line network traffic change detection system on a NetFPGA platform. Several different universal hash functions are implemented for comparison on latency and hardware resource consumption. The device resource utilization of the design (H=3, K=32k) is shown in Table 1. We stress the system by the IXIA traffic generator. The system hardware is capable of detecting the significant change of network traffic up to 4Gbps line rate with low error rate. Trace driven experiments are conducted and verified with the simulation results, shown in Table 2.

We aim for further optimizing the system and integrating with a router or switch design where actions can be proceeded with the detections. The design can also include functions such as adjusting observing epoch dynamically for fast response and sending out observing sketch to a centralized analyzer where linear sketch processing can be conducted for global awareness of networking traffic.

7. ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for their valuable comments and suggestions. This work is supported in part by National Science Council grants NSC 98-2221-E-033-025.

8. REFERENCES

[1] RFC 3917 - requirements for IP flow information export (IPFIX). http://tools.ietf.org/html/rfc3917.

[2] N. Alon et al. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] D. Barman, P. Satapathy, and G. Ciardo. Detecting attacks in router using sketches. In *Workshop on High Performance Switching and Routing, HSPR '07*, June 2007.

[4] D. Barman, P. Satapathy, and G. Ciardo. detecting attacks in routers using sketches. In *High Performance Switching and Routing, 2007. HPSR '07. Workshop on*, pages 1–6, 2007.

[5] T. Bu, A. Chen, and P. P. C. Lee. A fast and compact method for unveiling significant patterns in high speed networks. *IN PROC. OF IEEE INFOCOM*, 2006.

[6] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, Apr. 1979.

[7] Charikar et al. Finding frequent items in data streams. *TCS: Theoretical Computer Science*, Vol.312:3–15, 2004.

[8] G. Cormode and M. Hadjieleftheriou. Finding the frequent items in streams of data. *Commun. ACM*, 52(10):97–105, 2009.

[9] G. Cormode and S. Muthukrishnan. Improved data stream summaries: The count-min sketch and its applications. Technical Report 2003-20, DIMACS, June 2003.

[10] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 296–306. ACM, 2003.

[11] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *IEEE INFOCOM*, 2004.

[12] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. *Proceedings of 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004*, 3:1534–1545, 2004.

[13] A. Das, D. Nguyen, J.Zambreno, G. Memik, and A. Choudhary. An fpga-based network intrusion detection architecture. *IEEE Transactions on Information Forensics and Security*, 3(1):118–132, March 2008.

[14] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the 2007 workshop on Large scale attack defense*, pages 145–152, Kyoto, Japan, 2007. ACM.

[15] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 159–171, Marseille, France, 2002. ACM.

Table 3: Latency and resource utilization of various of standalone hash functions (32-bit key). The pipelined implementation is capable of processing a 32-bit key at each UDP clock cycle.

Hash Function	Latency	Resources	XC2VP50 Utilization	Percentage
2-Universal (Tabulation)	8ns	Slices	105 out of 23,616	0%
		4-Input LUTs	183 out of 47,232	0%
		I/O Blocks	69 out of 692	9%
2-Universal (Pipelined Multiplier)	56ns	Slices	4,122 out of 23,616	3%
		Slice Flip Flops	7,164 out of 47,232	3%
		4-Input LUTs	4,963 out of 47,232	2%
		I/O Blocks	70 out of 692	10%
		MULT18X18s	72 out of 232	10%
4-Universal (Tabulation)	32ns	Slices	5616out of 23,616	24%
		4-Input LUTs	10,090out of 47,232	21%
		I/O Blocks	68 out of 692	10%
4-Universal (Pipelined Multiplier)	192ns	Slices	902 out of 23,616	17%
		Slice Flip Flops	1,590 out of 47,232	15%
		4-Input LUTs	1,049 out of 47,232	10%
		I/O Blocks	70 out of 692	10%
		MULT18X18s	24out of 232	31%

[16] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better NetFlow. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 245–256, Portland, Oregon, USA, 2004. ACM.

[17] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[18] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[19] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Databases*, pages 180–191. VLDB Endowment, 2004.

[20] B. Krishnamurthy et al. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement (IMC-03)*, pages 234–247, New York, Oct. 27–29 2003. ACM Press.

[21] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247, Miami Beach, FL, USA, 2003. ACM.

[22] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 147–152, Rio de Janeriro, Brazil, 2006. ACM.

[23] MAWI. MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/.

[24] MAWI. MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/samplepoint-B/20030227/200302270000.html.

[25] S. Muthukrishnan. Data streams: algorithms and applications. In *SODA '03: Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 413–413, 2003.

[26] S. Muthukrishnan, E. van den Berg, and Y. Wu. Sequential change detection on data streams. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 551–550. IEEE Computer Society, 2007.

[27] D. Nguyen, G. Memik, and A. Choudhary. Real-time feature extraction for high speed networks. In *Proceedings of International Conference on Field Programmable Logic and Applications 2005*, pages 438–443. IEEE, 2005.

[28] S. Pati, R. Narayanan, G. Memik, A. Choudhary, and J. Zambreno. Design and implementation of an fpga architecture for high-speed network feature extraction. In *Proceedings of International Conference on Field-Programmable Technology, ICFPT 2007*, pages 49–56. IEEE, 2007.

[29] M. V. Ramakrishna, E. Fu, and E. Bahcekapili. A performance study of hashing functions for hardware applications. *IN PROC. OF INT. CONF. ON COMPUTING AND INFORMATION*, pages 1621—1636, 1994.

[30] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 207–212, Taormina, Sicily, Italy, 2004. ACM.

[31] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M. Kao, and G. Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Trans. Netw.*, 15(5):1059–1072, 2007.

[32] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 615–624, New Orleans, Louisiana, 2004. Society for Industrial and Applied Mathematics.

[33] A. Turner. Tcpreplay Suite. http://tcpreplay.synfin.net/trac/.