NC STATE UNIVERSITY

High-Throughput Sketch Update on a Low-Power Stream Processor

Yu-Kuen Lai
Dept. of Electrical Engineering
Chung-Yuan Christian Univ.
Chung-Li, Taiwan

Greg Byrd
Dept. of Electrical and Computer Engineering
Center for Embedded Systems Research
NC State University
Raleigh, NC

CESR

Outline

- ✓ Motivation
- ✓ Data Stream Processing & Imagine Stream Architecture
- ✓ The Sketch Data Structure
- ✓ Implementation and Performance
- ✓ Conclusion & Future Work

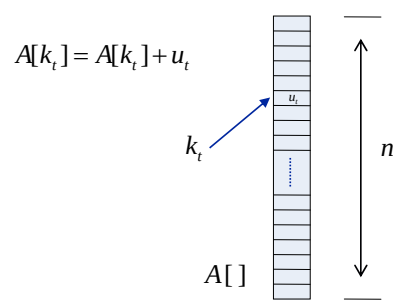
Motivation

[Slide from "Querying and Mining Data Streams: You Only Get One Look", VLDB'02, by Garofalakis et al.]

- ✓ **A growing number of applications are operated base on streams of data**
  - Traffic measurement & analysis for
    - Infrastructure planning
    - Capacity forecasting accounting
    - Security related
- ✓ **Application characteristics**
  - Massive volumes of data (several terabytes)
  - Data arrives at a rapid rate
- ✓ ***How to process queries and compute statistics on data streams in real-time?***

Data Stream Model

- ✓ **A data stream is a massive sequence of elements**
  - An input stream $\phi = (a_1, a_2, ...)$ arrives sequentially, item by item
  - Each item $a_t = (k_t, u_t)$ consists of a key and an update
    - key space $k_t \in \{0, ..., n-1\}$
    - Update,
  - A time varying **signal** A[ ] (an array of n buckets)
  - The arrival of each item cause the **signal** **A[]** to be updated

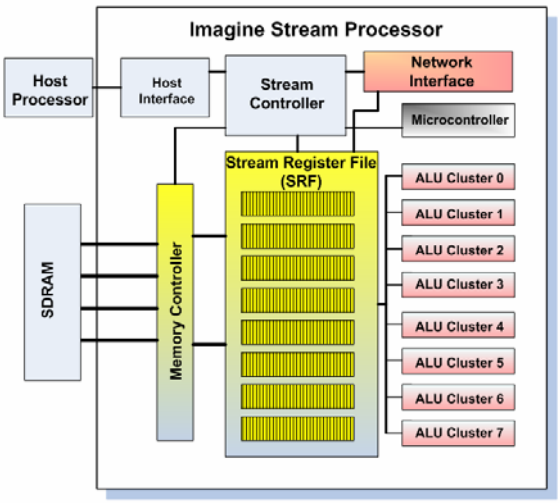


What Do We Need?

- ✓ **Fast Counter Update**
  - A smaller but faster SRAM
- ✓ **Being Programmable**
  - Accommodate different algorithms
  - Different threshold/approximate accuracy for different applications
- ✓ **Approximated Answers**
  - May not be able to produce exact answers due to limited resources (memory and processing power)
- ✓ **High Computation Power**
  - The statistic operations are computational intensive

Imagine Stream Processor

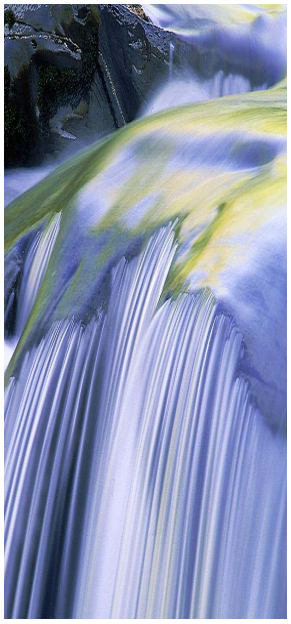
- ✓ **Built by Dr. William J. Dally's research team at Stanford University**
- ✓ **Designed for stream media processing**
- ✓ **A VLIW programmable co-processor supports stream programming model in SIMD fashion.**
- ✓ **Has 3 Levels of memory hierarchies**
- ✓ **Key modules are organized around the Stream Register File (SRF) through stream buffers**



Stream Processing

✓ What is Stream?

- An important data representation in stream programming model
- A collection of data records of variable length.
- Streams are inputs to kernels where computation is performed on its elements.



Stream

www.etplanet.com

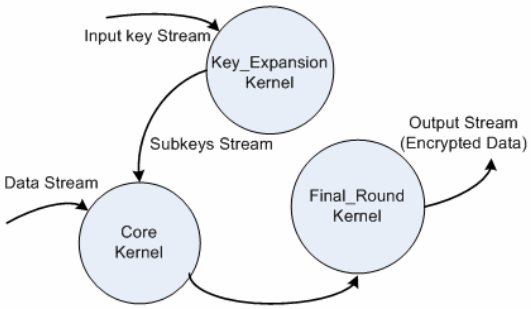
Programming Model

✓ Kernel Level

- Computations are done locally within the kernel
- Operates on streams as inputs and produces streams as outputs
- No arbitrary memory reference
- Loops are the ONLY control flow operations
- Conditional=Predicates

✓ Stream Level

- Run at the Host processor
- Kernel invocation
- Orchestrating the flow of streams



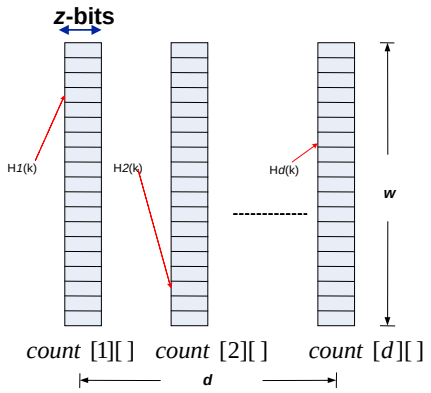
Some Performance Highlights

- ✓ **AES (2Gbps)**
  - ✓ Advanced Encryption Standard, Rijndael
  - ✓ In ECB and OCB modes with key agility
  - ✓ Our best/worst case is 32/76 cycles, 41 cycles for the variable sized packet (AIX-1054837521-1)
- ✓ **MMH Message Authentication (7Gbps)**
  - ✓ Multilinear Modular Hashing
  - ✓ Achieving Multi-Gigabit throughput. Ultra fast and *unconditionally* secure.
  - ✓ No matter how much computing power the adversary has. The probability for the adversary to compromise is lower than a probability *p*.
- ✓ **Bloom filter based Content Inspection Engine (400Mbps)**
  - Matching 2000+ signatures up to 400 Mbps with 500Mhz system clock. (1500 bytes packet)
  - The false positive error rate is 9.8e-7

The Count-Min Sketch

[Cormode & Muthukrishnan, Dec 2003]

- ✓ ***It's a probabilistic, approximated*** **algorithm**
- ✓ **The BEST existing sketch scheme** [2005 G. M. Lee et al.]
- ✓ **The operation is based on a two-dimensional array, count[d][w]**



The Count-Min Sketch (Cont.)

Update

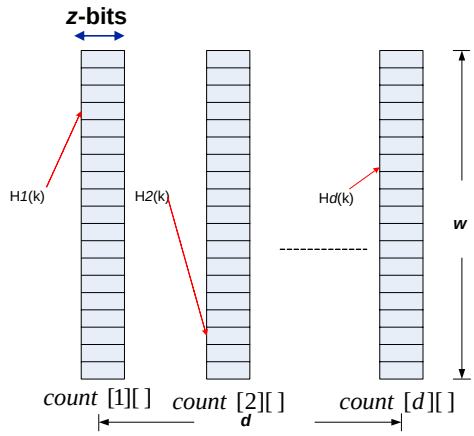
- ✓ For each arrived item $a_k = (k, u)$
- ✓ Hash the key by *d* different independent hash functions
- ✓ Use those *d* hash values as indexes to update the value u into each array

$$count[j][h_j(k)] += u$$
$$1 \le j \le d$$

Point Query Q(k)

- ✓ Given a key *k*, again we hash the key
- ✓ Use these hash values as indexes to look up the value stored in each array
- ✓ The answer to Q(k) is to pick the *minimum* of all these *d* values

$$\hat{a}_k = \min_j \{count[j][h_j(k)]\}$$
$$1 \le j \le d$$



Many Applications [Cormode & Muthu, SIAM ICDM05]

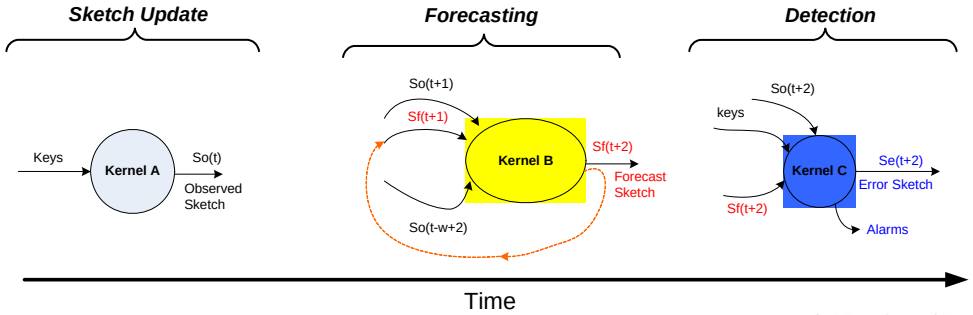
- ✓ **Significant differences** [Charikar et al, ICALP'02] **and relative changes** [Cormode & Muthu INFOCOM'04]
- ✓ **Anomaly Detections** [Krishnamurthy et al, SIGCOMM'03]
- ✓ **Heavy hitters** [Cormode & Muthu ACM PDS 03]
- ✓ **Top-K items** [Manku & Motwani ICDM'05]
- ✓ **Estimating frequent items** [Manku & Motwani ICVLDB'02]

Change Detection

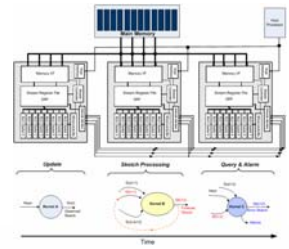
- ✓ **Monitoring the significant differences in traffic attributes over two observing intervals**
- ✓ **Attributes of interest**
  - Number of packets
  - Flows
  - Total bytes
- ✓ **Typical Approaches**
  - Brute Force (store & sort)
  - Sampling
  - **Sketch**

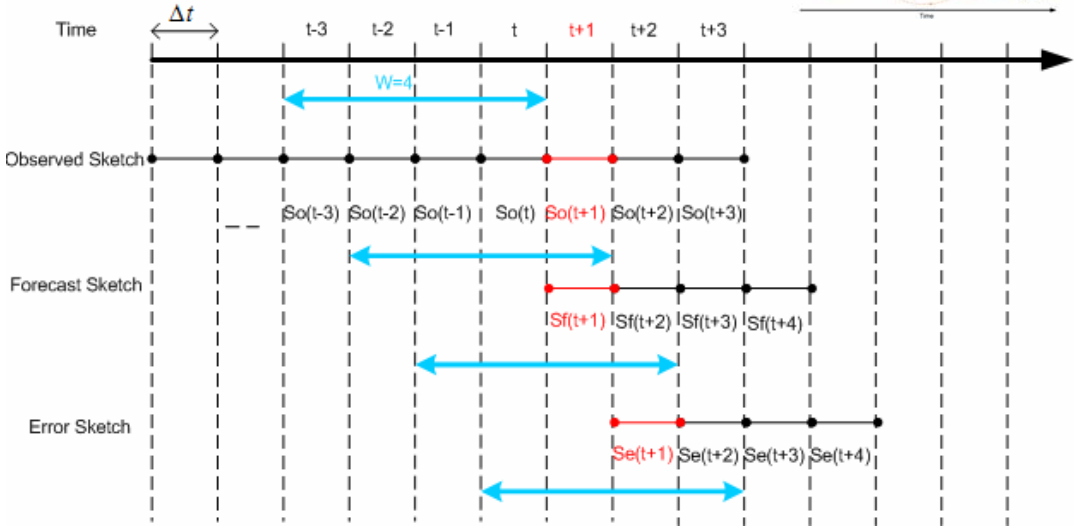
Sketch-based Change Detection

- ✓ **Absolute difference based on *K-ary sketch* by Krishnamurthy et al.**
- ✓ **K-ary sketch**
  - Same sketch update process
  - Require 4-universal hash function
  - Different point query procedure
- ✓ **Three major modules:**
  - Sketch update
  - Forecasting
  - Detection



Sketch-based Change Detection (cont.)

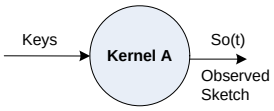


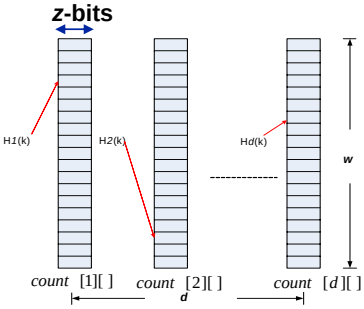


The Update Module

Sketch-based Change Detection (cont.)

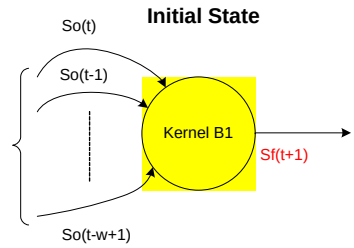
- ✓ **Same update procedure as that in the CM sketch**
- ✓ **Updating keys for a time interval** Δt
- ✓ **The update has to be quick enough to process the incoming packets at line rate**
  - For every packet
    - Key extraction
    - Hashing
    - Updating the counters
  - It's 32ns for processing a minimum-sized IP packet at 10Gbps





The Forecast Module

Sketch-based Change Detection (cont.)

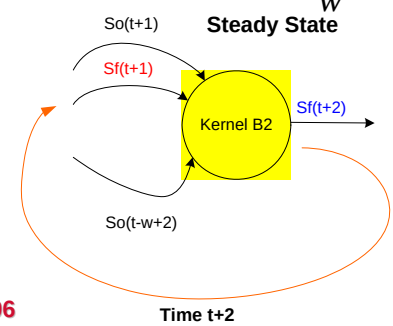


- ✓ **Forecast model based on the *moving average***
- ✓ **The forecast sketch**

$$S_f(t+1) = \frac{1}{w}\sum_{i=0}^{w-1} S_o(t-i)$$

- ✓ **Compute the forecast sketch incrementally**

$$S_f(t+2) = S_f(t+1) + \frac{1}{w}(S_o(t+1) - S_o(t-w+1))$$



The Change Detection Module

Sketch-based Change Detection (cont.)

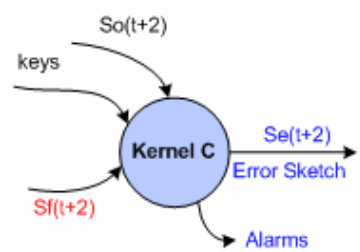
- ✓ **Alarm threshold** T_A

$$T_A = T \cdot [Estimate\_F_2(S_e(t))]^{1/2}$$

$$S_e(t) = S_o(t) - S_f(t)$$

$$Estimate\_F_2(S_e(t)) = median_{i \in H}\{F_2^{h_i}\}$$

$$F_2^{h_i} = \frac{k}{k-1}\sum_{j \in |k|}(T_s[i][j])^2 - \frac{1}{k-1}(sum(S))^2$$



- ✓ **Given a key, Raise the alarm if**

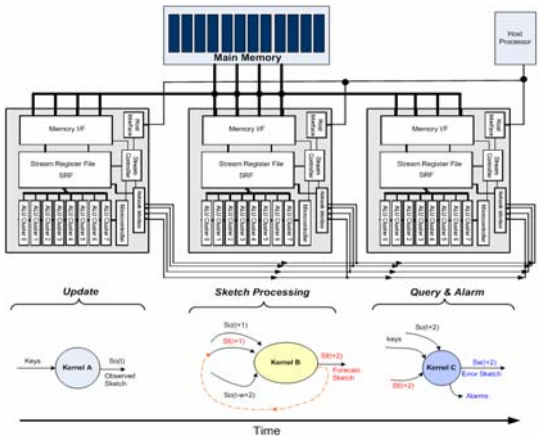
$$Estimate(S_e(t), key) > T_A$$

$$Estimate(S_e(t), key) = median_{i \in |H|}\{\frac{T[i][h_i(key)] - sum(S)/k}{1 - 1/k}\}$$

The Bottleneck

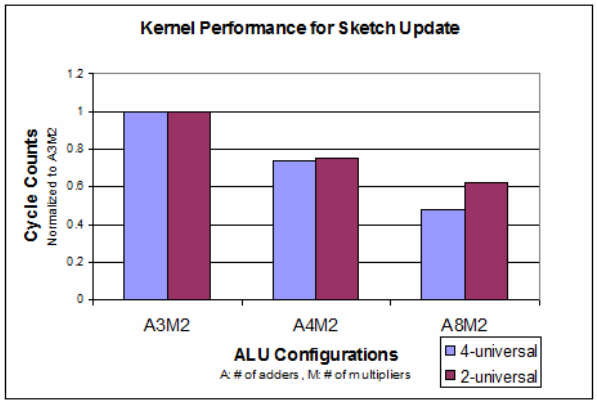
Sketch-based Change Detection (cont.)

- ✓ **The sketch processing (forecasting and Detection) is based on a time interval Δt**
  - Usually the interval is set in minutes, say 1 or 5 minutes
  - Kernel B and C run once per Δt
- ✓ **The bottleneck is in the Sketch update module**



Sketch Update Performance

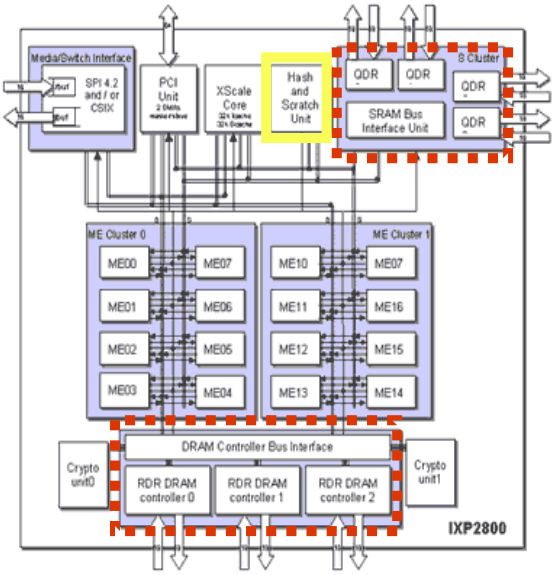
- ✓ **For hashing and update a 32-bit key**
  - 15 cycles (2-universal)
  - 33 cycles (4-universal)
- ✓ **It's 10.6Gbps and 4.8Gbps for processing 40-byte packets with system clock of 500 MHz**



Intel IXP2800 Network Processor Architecture

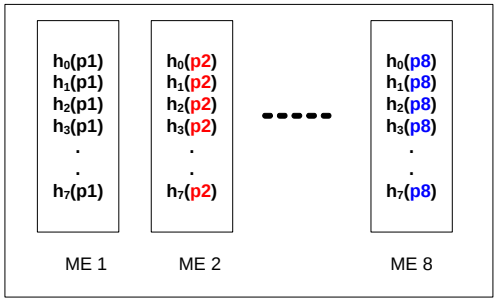
Memory Hierarchies

- ✓ **Local Memory (3)**
  - 2,560 bytes
- ✓ **Scratchpads (60)**
  - 16K Bytes
- ✓ **SRAM (150)**
  - 4 Channels, each supports 4GBps @250MHz
- ✓ **DRAM (300)**
  - Total size of 2G Bytes
  - Bandwidth 50Gbps



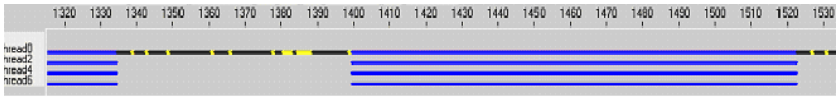
Sketch Update On IXP2800

- ✓ **Decisions**
  - For 1,024 x 8, 32-bit counter array
    - Local Memory of 2,560 bytes is too small to hold the array
    - Scratchpad of 16K bytes is too small
    - DRAM has ~300 cycles of access latency
  - SRAM is the choice
- ✓ **Arrangement**
  - Each thread in the ME fetch the packet header and calculate 8 hashes

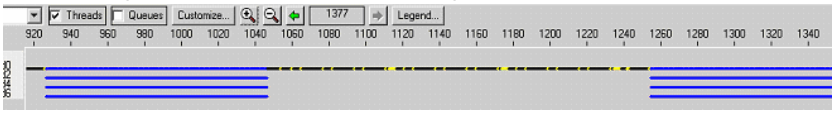


Performance

- ✓ **Atomic Access**
  - Sram-test-and-add ()
  - ME issues a ***single*** command with address and update value
  - **120 cycles** latency for memory access
- ✓ **2-Universal Hashing (single thread)**
  - 70 cycles to hash a 32-bit key



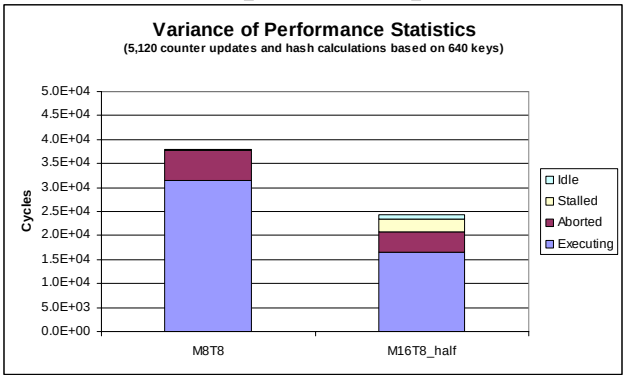
- ✓ **4-Universal Hashing (single thread)**
  - 220 cycles to hash a 32-bit key



Performance (cont.)

- ✓ **Improvement on shortening the computation?**
  - Hardware assists
    - ♦ CRC unit may not provide the quality needed
    - ♦ Bus Arbitration overhead on share resources (hash on SHaC unit)
  - Table lookup
    - ♦ Large table is needed
      - Three 64K x 4 bytes needed for tabulation of 4-universal hash function
    - ♦ May introduce more memory access latency
  - **Still bounded by memory access latency**
- ✓ **Hide the latency with more threads**

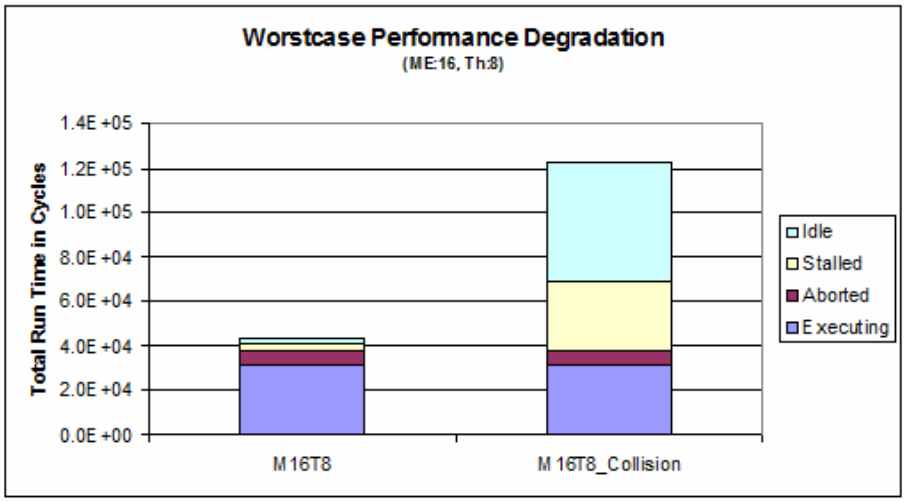
Performance (cont.)



- ✓ Double the amount of MEs, 16 instead of 8
- ✓ Same number of hash calculations and counter updates
- ✓ 36 % decrease in total cycles

Worst-case Scenario

- ✓ Collisions due to hash and update on 1,280 keys of the same source IP address



Comparison

	Imagine	IXP2800
Throughput 40-byte packet	10.86Gbps	13.34Gbps
Avg. Hash&Update (2-universal)	3.68ns	3ns
Clock	500MHz	1,400MHz
Hardware	8 Clusters	16 MEs
Typical Power	2.2W	21~26W
Max Power	4W	30W
Technology	TI 0.15um	Intel 0.13um

Some Performance Highlights

- ✓ Sketch Update
  - ✓ 10Gbps (2-universal)
  - ✓ 4.8Gbps (4-universal)
- ✓ AES (2Gbps)
- ✓ MMH Message Authentication (7Gbps)
- ✓ Bloom filter based Content Inspection Engine (400Mbps)

Conclusion

- ✓ **Low power consumption**
- ✓ **High computation power**
- ✓ **Flexibility**
  - The data structure can be served as the basis for various networking applications
  - Capable of accommodating many dynamic aspects
    - Adjusting thresholds
    - Change statistical models
    - Change methodologies due to accuracy requirements

Future Work

- ✓ **Sketch/Stream Manipulations**
  - For better accuracy
- ✓ **Integrate the simulation framework**
  - Construct the network interfaces at Media Access layer (MAC)
  - Facilitate the exploration of many network system designs
- ✓ **Multi-SIMD hybrid architecture**
  - A group of SIMD clusters share the same microcontroller issuing the instructions while different group of these entities behave in MIMD mode