UnivMon: Software-defined Monitoring with Universal Sketch

Zaoxing (Alan) Liu

Joint work with Antonis Manousis (CMU), Greg Vorsanger (JHU), Vyas Sekar (CMU), and Vladimir Braverman (JHU)
Network Management: Many Monitoring Requirements

- Traffic Engineering
- Worm Detection
- Anomaly Detection
- Accounting

- "Flow size distribution"
- "SuperSpreaders"
- "Heavy-hitters"
- "Entropy"
- "Traffic Changes"

SDN Controller (OpenDayLight etc.)
Network Management: Many Monitoring Requirements

- Traffic Engineering
- Anomaly Detection
- Worm Detection
- Accounting
- “Flow size distribution”
- “Entropy”, “Traffic Changes”
- “SuperSpreaders”
- “Heavy-hitters”

Typical Measurement Questions:

- Who’s sending a lot more traffic than 10min ago? (Change)
- Who’s sending a lot from 10.0.1.0/16? (Heavy Hitter)
- Are you being DDoS-ed?
Example: A Victim being DDoSed
Traditional: Packet Sampling

Sample packets at random, aggregate into flows

Flow = Packets with same pattern Source and Destination Address and Ports

Flow reports

Estimate: FSD, Entropy, Heavyhitters …

Not good for fine-grained analysis
Extensive literature on limitations for many tasks!
Application-Specific Sketches

Heavy Hitter

Entropy

Superspreader

Bloom-filter, Count-min Sketch, reversible sketch, etc.

Packet Processing

Counter Data Structures

Counter Data Structures

Counter Data Structures

Traffic

Complexity: Need per-metric implementation

Recent Example: OpenSketch [NSDI’13]

Trend: Many more applications appear!
A Generic Method

1. Generic
2. Not too expensive
3. Accurate for fine-grained analysis
Outline

• Motivation

• UnivMon System Design

• UnivMon Algorithm

• Evaluation
Our Design: UnivMon

Control plane
- Metric estimation
- Monitoring manifest distribution

Data Plane
- Traffic data collection
- Sketch counter updates

1. Distribute Manifests
- Sketches, Dimension, Memory

2. Collect Sketch Counters

3. Metric Estimation
- Manifest computation

- Topology Routing

- Late-binding for applications: data plane is general-purpose.
- One Sketch: no need of memory allocation for multiple tasks
Outline

• Motivation
• UnivMon Design
• UnivMon Algorithm
• Evaluation
Theory of Universal Streaming \([\text{BO’10, BO’13}]\)

Estimated
G-sum

‘Universal’
Sketch

As long as \(g(f_i)\) does not grow monotonically faster than \(f_i^2\)

\[
G\text{-sum} = \sum_{i=1}^{n} g(f_i)
\]

frequency vector is \(< f_1, f_2, \ldots, f_n >\)

(A stream of length \(m\) with \(n\) unique items)

- When \(g(x) = x^0\), G-sum is \# of distinct items. [SuperSpreader]
- When \(g(x) = x\log(x)\), G-sum is the entropy norm. [Entropy]
Basics: Streaming Algorithms

(A stream $S$ of length $m$ with $n$ unique items)

- **K-th Frequency Moments:**

  $$F_k(S) = \sum_{i=1}^{n} f_i^k$$

  - $F_0$ is $n$: the number of distinct items in $S$.
  - $F_1$ is $m$: the length of $S$.
  - $F_2$ is Gini's index of homogeneity.

  When $k>2$, the space lower bound is $\Omega(n^{1-2/k})$ [CKS’03]
  (For current applications, the case of $k>2$ is not that interesting)

- **Heavy Hitters (Frequent Items):**

  $g$-heavy item $i$ : $g(f_i) > \alpha$ G-sum for some $\alpha$

  Count-min sketch [CM’04] is a popular L1-heavy hitter algorithm
Universal Sketch Data Structure

Levels 0

In Parallel

0

1 1 5 1 3 3 1 2 4 6 5

H₁(1)=1, H₁(5)=1, H₁(2)=1

1

1 1 5 1 1 2

H₂(5)=1, H₂(2)=1

... 5 2 5

H₃(2)=1

log(n) 2

Generate log(n) substreams by zero-one hash funcs H₁....Hₙ

L2 Heavy Hitter Algorithms

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+4</td>
<td>+2</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>-4</td>
<td>+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heavy Hitter Alg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Parallel

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+4</td>
<td>-2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+4</td>
<td>-2</td>
<td>+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>-2</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heavy Hitter Alg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Parallel

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>-2</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>-2</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heavy Hitter Alg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Parallel

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
</tr>
<tr>
<td>-1</td>
</tr>
</tbody>
</table>

Count Sketch Alg

Heavy Hitters

(1,4), (3,2),(5,2)

Heavy Hitters

(1,4), (5,2),(2,1)

... ...

Heavy Hitters

(5,2), (2,1)

(2,1)

Count-Sketch etc. Similar to counting bloom filter
Estimating G-sum

Counters from Universal Sketch

Levels

0

(1, 4), (3, 2), (5, 2)

(1, g(4)), (3, g(2)), (5, g(2))

1

(1, 4), (5, 2), (2, 1)

(1, g(4)), (5, g(2)), (2, g(1))

..."}

log(n)

(2, 1)

(2, g(1))

Y_0 = 2g(1) + 2g(2) + g(4)

A_1, ply arbitrary g()

Y_1 = g(1) + g(2) + g(4)

Recursive Steps:

Y_{i-1} = 2Y_i + new counters − repeated counters
Intuitions of Universal Sketch

1. Group items into log(n) groups
2. Find (≥) one from each group
3. Recursively sum them all!

Idea: Detect items that contribute most to G-sum
Putting it together: UnivMon

Hash functions and sketch counters

Offline Computation

Control Plane

Data Plane

m incoming pkts

O(log(n)) substreams

Paralleled instances

Heavy Counters

CountSketch

App. Metrics

App.-specific

Recursive steps

App. -specific
I want to monitor: Heavy Hitter, DDoS, Entropy, etc.

OK!

OK!

OK!

OK!

OK, I got it. But results are not good enough!

Packet Sampling Solution

Flow reports

Application-specific Approach

I need Count Min Sketch for HH

I need additional bitmap for DDoS

I need k-ary sketch for Changes

I need algorithm xxx for xxx. Oh no!

Our Approach: UnivMon

Controller

I will calculate a large class of funcs!

I need more applications!
Outline

• Motivation

• UnivMon Design

• UnivMon Algorithm

• Evaluation
Key Evaluation Questions

• Feasibility of hardware deployment?
  ✓ Removed expensive data structures
  ✓ Implemented in P4

• UnivMon’s accuracy
  ✓ Compare to the up-to-date sketch algorithms

• UnivMon’s stability
  ✓ Stabilities over different traces
Evaluation

• Comparison with custom sketches via OpenSketch
Evaluation

- Comparison with custom sketches via OpenSketch
Conclusions

• Traditional packet sampling based approaches have limitations for fine-grained analysis.

• Custom Sketches (e.g. OpenSketch)
  • Need to know applications beforehand
  • High Implementation cost
  • More expensive on multi-tasks

• UnivMon: an efficient and general sketching based approach with late-binding on applications.
Future Directions

• Multi-dimensional data

• Performance optimization for hardware

• Dynamically change monitoring scope

• New Theories on Universal Sketch:
  ▪ Sliding Window Model [SODA’16]
  ▪ Functions of One Variable [PODS’16]
  ▪ Symmetric Norms [arXiv 1511.01111]
Thank you!