Scalable and Robust DDoS Detection via Universal Monitoring

Vyas Sekar
Joint work with:
Alan Liu, Vladimir Braverman JHU
Hun Namkung, Antonis Manousis, CMU
DDoS attacks are getting worse

Increasing in number
Increasing in power
Increasing in diversity

DDoS Attacks Cost $40,000 Per Hour
Incapsula, 11/12/2014

FBI WARNS OF INCREASE IN DDoS EXTORTION SCAMS
Threatpost, 7/31/2015

China Appears to Attack GitHub by Diverting Web Traffic
The New York Times, 3/30/2015

Half of companies experience more than five DDoS attacks a year.
Neustar, 2014

The DDoS That Almost Broke the Internet
Cloudflare, 3/27/2013

Wave of 100Gbps 'mega' DDoS attacks hits record level in 2014
Techworld, 7/16/2014

NTP ATTACKS: Welcome to The Hockey Stick Era
Arbor Networks, 2/14/2014

Tsunami SYN Flood Attack
Radware, 10/7/2014

©2017 Open-NFP
Many attacks, many algorithms!

- SYN Flood
- UDP Flood
- NTP Flood
- DNS Amplification

- Who’s sending a lot more traffic than 10min ago?
- Who’s sending a lot to 10.0.1.0/16?
- Is there asymmetry in packet counts in directions?
Sample packets at random, group into flows

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

Prior work: Not good for fine-grained analysis!

Estimate: FSD, Entropy, Heavy Hitters …
Alternative: App-specific sketches

**Heavy Hitter**
- Application-Level Metric
- Counter Data Structure
- Packet Processing

**Entropy**
- Application-Level Metric
- Counter Data Structure
- Packet Processing

**Superspread**
- Application-Level Metric
- Counter Data Structure
- Packet Processing

Pre-deployed Algorithms

Traffic

*Higher Complexity* with more applications

*Higher development time* as new applications appear

*Tight Binding* between monitoring data and control plane
Driving question for our work

Generality
Late Binding

e.g., NetFlow

Today

XOR

Fidelity

e.g., Sketches

AND

?
Many open questions..

- Does such a construction exist?
- Does it extend to a network-wide setting? e.g., Multiple paths, Multiple dimensions
- Is it competitive w.r.t. custom algorithms?
- Is it feasible to implement?
Roadmap for this talk

- Does such a construction exist?
- Does it extend to a network-wide setting? e.g., Multiple paths, Multiple dimensions
- Is it competitive w.r.t. custom algorithms?
- Is it feasible to implement?
Concept of Universal Streaming

• Basic Streaming Algorithms:
  (A stream of length \( m \) with \( n \) unique items)
  \[
  \begin{bmatrix}
    1 & 1 & 5 & 1 & 3 & 3 & 1 & 2 & 4 & 6 & 5
  \end{bmatrix}
  \ldots
  \]

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

  Frequency Moments \( F_k = \sum_{i=1}^{n} f_i^k \)

  \( F_2 \): AMS Sketch, Count Sketch

  \ldots

  One algorithm solves one problem

• Universal Streaming?
  \[
  \begin{bmatrix}
    1 & 1 & 5 & 1 & 3 & 3 & 1 & 2 & 4 & 6 & 5
  \end{bmatrix}
  \ldots
  \]

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

  Universality: arbitrary \( g() \) function?

  G-sum = \( \sum_{i=1}^{n} g(f_i) \)
Thm 1: There exists a universal approach to estimate G-sum when \( g() \) function is non-decreasing such that \( g(0)=0 \), and \( g(f \downarrow i) \) doesn’t grow monotonically faster than \( f \downarrow i^2 \).

Thm 2: A universal sketch construction can be used to estimate G-sum with high probability using polylogarithmic memory.
Intuition behind Universality

Informal Definition: Item $i$ is a $g$-heavy hitter if changing its frequency $f_i$ significantly affects its G-sum.

Case 1: there is one sufficiently large a $g$-heavy hitter

Most of mass is concentrated in this heavy hitter. Use L2 Heavy Hitter algorithm to find such a heavy hitter.

Case 2: there is NOT single sufficiently large a $g$-heavy hitter

Find heavy hitters on a series of sampled substreams of increasingly smaller size.
Universal Sketching Algorithm

Generate $\log(n)$ substreams by zero-one hash functions $H_1, \ldots, H_{\log(n)}$

Level In Parallel

0 1 1 5 1 3 3 1 2 4 6 5

1 1 1 5 1 1 2 5

$\ldots$ 5 2 5

$\log(n)$ 2

Count-Sketch etc.

Heavy Hitter Alg

Heavy Hitter Alg

Heavy Hitter Alg

Heavy Hitter Alg

Heavy Hitters and Counters

Estimated G-sum

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

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

(5,2), (2,1)

(2,1)
Universal Monitoring Realization

Application-specific Computation

Sampling (Hash func)

Update Counters

Traffic

Sampling → Sketching → Top-K → App-Estimation

Top-K

Possible keys

Sketching Registers

App 1

......

App n
Roadmap for this talk

Does such a construction exist?

Does it extend to a network-wide setting?  
e.g., Multiple paths, Multiple dimensions

Is it competitive w.r.t. custom algorithms?

Is it feasible to implement?
Network-Wide Problem

N nodes
D dimensions
(e.g., src, srcdst)

Trivial sol: place D*N sketches
Our goal: Place s sketches, where s<<D*N
One-big-switch abstraction
Roadmap for this talk

- Does such a construction exist?
- Does it extend to a network-wide setting? e.g., Multiple paths, Multiple dimensions
- Is it competitive w.r.t. custom algorithms?
- Is it feasible to implement?
Evaluation Setup

- Traces: CAIDA backbone traces
  - Split into different “epoch” durations

- Memory setup: 600KB—5MB

- Application metrics: HH, Change, DDoS

- Custom algorithms from OpenSketch
UnivMon is Competitive

Key Takeaways:
- Stable cross traces
- Error gap < 3.6%
- Good accuracy with limited memory

UnivMon (Total 600KB)
OpenSketch (600KB/task)
UnivMon is Better as Portfolio Grows!

![Graph showing error gap percentage for different app sets and configurations.](image)

- **Appset1**
  - {HH}
  - Error Gap: [-10, 10]

- **Appset2**
  - {HH, DDoS}
  - Error Gap: [-10, 10]

- **Appset3**
  - {HH, DDoS, Change}
  - Error Gap: [-10, 10]

Legend:
- Red: Heavy Hitter
- Blue: DDoS
- Yellow: Change Detection
Does such a construction exist?

Does it extend to a network-wide setting? e.g., Multiple paths, Multiple dimensions

Is it competitive w.r.t. custom algorithms?

Is it feasible to implement?
Mapping Data Plane to P4

1. Sampling
   - P4 Hash Funcs

2. Sketching
   - P4 Registers

3. Top-K
   - Hash Funcs + P4 Registers

Custom Libraries

App-Estimation
Design choices for realization

Sampling → Sketching → Top-K

App-Estimation

HW Complexity (need Priority Queue)
Storage/Comm Overhead (report Top-K to controller)

Hard in hardware

Sampling → Sketching → Top-K

App-Estimation

HW Complexity
Storage/Comm Overhead (report entire sketch/keys)

Several MBs more
Implementation in Netronome: Step 1

Initial attempt: We tried with UnivMon P4 Code

Found out limitation of P4

- No Loop statement – out of space in Netronome
- Lack of Expressiveness, want to store seed values for hash. Store this at low level memory.
Implementation in Netronome: Step 2

So we switched to Micro-c capabilities

Some difficulties in porting/understanding APIs figuring out performance bottlenecks

We used the simulator to profile the bottleneck

Found out hash computation is the problem!
• Shift operation instead of modular operation
  • \(a, b: 64\) bit random integer, \(x: 32\) bit key
  • \(H_{a,b}(x) = ((ax + b) \% p) \% m\)
  • \(H_a h(x) = ((ax + h) >> 32) \& n\)
• Shift operation is much faster than modular operation in Netronome

• UnivMon can exploit parallelism with Netronome. Atomic engine did a great job to solve synchronization issues with sketch counters

• Limitation: Shift operation can’t guarantee enough randomness of hash functions and fair accuracy of sketching
Ideas: Use Tabular Hash

- Memory read is faster than modular operation and it has higher independence

<table>
<thead>
<tr>
<th></th>
<th>32bit</th>
<th>64bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Two Indep</td>
<td>CW trick 32</td>
</tr>
<tr>
<td>Independent</td>
<td>2-independent</td>
<td>5-independent</td>
</tr>
<tr>
<td>Key Idea</td>
<td>(ax+b) % p % m</td>
<td>multiplication &amp; shift</td>
</tr>
<tr>
<td>Memory</td>
<td>0</td>
<td>20KB</td>
</tr>
<tr>
<td>Instructions</td>
<td>53 (8 multiplication)</td>
<td>37</td>
</tr>
<tr>
<td>Memory lookup</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Memory Needed for Netronome (X 27)</td>
<td>540KB</td>
<td>10MB</td>
</tr>
<tr>
<td>Implemented</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

2.7GB runs out of memory
204MB is possible to implement
Now all of tables are in the DRAM of NIC
### Tabular hash results - Kpps

<table>
<thead>
<tr>
<th>Line</th>
<th>Rate with max packet size (1500) : 3245</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netronome Performance - Kpps with min packet size (64)</td>
<td></td>
</tr>
</tbody>
</table>

- **Baseline (forwarding)** max pps: 17300K pps
- **Line Rate with max packet size (1500)**: 3245
Tabular hash throughput results

- Char Table 32 -> can cover 2 dimension with line rates
- 64bit is slower
Some lessons and takeaways

Simulator helped!
We could profile the bottleneck of our implementation with built-in simulator of IDE

UnivMon is feasible on NFP at line-rate
with 3 dimensions and 5-independent hash function

C programming with Netronome has greater flexibility!
Ongoing and Next Steps

APIs to write applications and queries on UnivMon

Suite of DDoS detection applications

Continue profiling and benchmarking

Other platforms as well (e.g., openvswitch, fd.io)
Conclusions

• DDoS Detection needs more flexibility and programmability

• Today: General XOR Flexible
  Vision: General + Flexible via Universal Monitoring

• Initial promise: Feasible, accurate, possible to implement

• Ongoing and future work:
  Performance profiling, “Northbound” APIs etc.