NitroSketch: Robust and General Sketch-based Monitoring in Software Switches

Alan (Zaoxing) Liu

Joint work with Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar
Need for network telemetry in software switches

Where: Virtual switches, White-box switches, DPDK, FD.io,...
Need for network telemetry in software switches

Where: Virtual switches, White-box switches, DPDK, FD.io …..

Traffic Engineering
“Flow Size Distribution”

Anomaly Detection
“Entropy”, “Traffic Changes”

Worm Detection
“SuperSpreaders”

Accounting
“Heavy Hitters”

Requirements: Performance, Accuracy
Sketches appear to be a promising alternative

<table>
<thead>
<tr>
<th>Flow key k</th>
<th>5</th>
<th>+1</th>
<th>7</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>+1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>+1</td>
<td>11</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>+1</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- Require **small** memory with **bounded** error rates.
- Guaranteed fidelity with arbitrary workloads.

Packet

- Require small memory with bounded error rates.
- Guaranteed fidelity with arbitrary workloads.
Reality: Sketches in software switches not performant!

- Single core 100% CPU.
- Cannot meet high line-rates: < 10Gbps
Existing proposals to speed up sketches?

- **SketchVisor** [SIGCOMM'17]:
  
  Performance increase (still < 10 Mpps)
  
  X Not robust on arbitrary workloads.

- **Elastic Sketch** [SIGCOMM'18]:
  
  Performance increase (> 14.8 Mpps)
  
  X Not robust on arbitrary workloads.

- **R-HHH** [SIGCOMM'17]:
  
  Performance increase (up to 14.8 Mpps)
  
  X Not general towards many tasks.
NitroSketch in a nutshell

A software-switch based sketching framework that simultaneously has:

- **Performance**: line-rate (40G) with minimal CPU and memory.
- **Generality**: support a variety of measurement tasks.
- **Robustness**: accuracy guarantees for any workload.
NitroSketch Approach

• Systematically analyze the performance bottlenecks.

• Learn key insights from strawman solutions (sampling, one-level hash).

• Reformulate sketching for software from first principles
  - Tradeoff slight memory increase, Novel counter sampling rather than packet.
Outline for this talk

• Motivation
• Understanding bottlenecks
• Design Insights
  • Strawman ideas
  • Our proposals
• Evaluation
• Conclusions and future work
Outline for this talk

- Motivation
- Understanding bottlenecks
- Design Insights
  - Strawman ideas
  - Our proposals
- Evaluation
- Conclusions and future work
Bottleneck Analysis

- Performance benchmarks using Intel VTune Amplifier.
- Hotspots for UnivMon [SIGCOMM’16].

<table>
<thead>
<tr>
<th>Func/Call Stack</th>
<th>Description</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxhash32</td>
<td>hash computations</td>
<td>37.29%</td>
</tr>
<tr>
<td>__memcpy</td>
<td>memcpy and counter update</td>
<td>15.91%</td>
</tr>
<tr>
<td>heap_find</td>
<td>heap operation</td>
<td>10.71%</td>
</tr>
</tbody>
</table>
Bottleneck B1: Many hash computations per packet

B1: Many (independent) hash computations per packet (~37% CPU)
Bottleneck B2: Many counter updates per packet

Packet → Hashing → +1 → +1 → +1 → +1 → Top keys

- d arrays of counters
- r counters per array

B2: Many counter updates (~15% CPU)
Bottleneck B3: Tracking keys is also expensive

B3: Expensive flow key data structure operations (e.g., heap (~10% CPU))
Outline for this talk

- Motivation
- Understanding bottlenecks
- Design Insights
  - Strawman ideas
  - Our proposals
- Evaluation
- Conclusions and future work
Outline for this talk

• Motivation
• Understanding bottlenecks
• Design Insights
  • Strawman ideas
  • Our proposals
• Evaluation
• Conclusions and future work
Strawman 1: Reduce hashes by using single array?

**Pros:** Simple idea to reduce hash computations.

**Cons:**
- Memory increase may cause cache misses.
- Even one lightweight hash per packet may be high!
Strawman 2: Reduce updates by packet sampling?

Flip a coin: with $p$

Pros: Simple idea to reduce hash/counter updates.

Cons:
- Memory increase may cause cache misses.
- Even one coin flip per packet may be high!
- Incurs accuracy-convergence tradeoff.
Key lessons from strawman solutions

• Can tradeoff memory for CPU reduction
  But need to ensure cache residency

• Sampling is promising
  But need to manage per-packet ops and convergence
How do we tackle this?

- We trade memory for CPU reduction while ensuring cache residency

- Sampling is promising
  But need to manage per-packet ops and convergence
Key Idea: Sample counter updates not packets!

Rethinking the way of doing sampling with sketches

Uniform Packet Sampling

Uniform Counter Sampling

Each Packet

Sketch

Sampled Packet
Key Idea: Sample counter updates not packets!

Multiple independent hashes key for memory efficiency

- Per-packet hash/counter updates can be reduced to **less than one**.
- Memory increase by $O(1/p)$: Much better than uniform packet sampling.
How do we tackle this?

• Can tradeoff memory for CPU reduction
  But need to ensure still cache resident

• Sampling works!
  but need to manage per-packet ops and convergence
Trick: Geometric sampling to reduce per-packet ops

Uniform sampling with $p$

Coin flip with $p$

Coin flip with $p$

Coin flip with $p$

Coin flip with $p$

Coin flip with $p$

Equivalent

Geometric sampling with $p$

Sampled.

Flip a coin with $p$

Interval is 4

Skipped

Sampled.

Flip a coin for next
Trick: Adapt sampling rate to manage convergence

- Sampling-based approach needs to receive enough packets before becoming accurate (need convergence).

- When packet rate isn't high,
  - we can sample more packets ⇒ faster converge

Adaptively adjusting sampling rate:
Packet rate: 1Mpps, sample with 1/1
Packet rate: 8Mpps, sample with 1/8
Packet rate: 64Mpps, sample with 1/64
NitroSketch: Putting it together

Update with +1/p
Select next 21
Skipped
Update with +1/p
Adjust the sampling rate when needed

Trade small space for speedup on CPU
NitroSketch is theoretically robust!

- NitroSketch offers accuracy guarantees for a variety of measurement tasks.

  Our theoretical analysis holds after receiving enough packets.

- In practice, we need ~2-4 Mil packets to converge.

*Check out our paper for more details!*
NitroSketch Inline Implementation

- Controller/Other Users
- User Space
  - Shared Buffer
  - OVS-DPDK
    - dpif-netdev forwarding
    - NitroSketchD
    - emc
    - dp
    - classider
    - ...
- Kernel Space
  - PMD
  - PMD
  - PMD
  - NIC
  - NIC
  - NIC

- Other versions: FD.io-VPP, BESS
NitroSketch achieves 40G on software switches

- Two threads with OVS-DPDK, VPP and BESS on Intel XL710 NIC.
- NitroSketch uses **no extra cores**.
NitroSketch can achieve higher throughput

- In-memory single thread (Intel E5 2620 v4 CPU)
- Algorithms use 5~10 independent hash functions
Guaranteed accuracy after convergence

- After received 2~4 Mil packets, Sketches achieve comparable (or better) accuracy as the original sketches.
NitroSketch outforms other solutions

- NitroSketch achieves higher accuracy when converged.
Conclusions

● Sketching is a promising alternative for software switch based telemetry.

● Performance of sketches is far from optimal.

● Existing efforts missing in performance, robustness, or generality.

● NitroSketch key ideas:
  Tradeoff small memory increase, Sample counters not packets, Geometric sampling to reduce packet operations, Adaptive sampling

● NitroSketch improves the performance of sketches by 1-2 orders of magnitude while retaining the robustness and generality

https://github.com/zaoxing/NitroSketch