Fast and Efficient Flow and Loss Measurement for Data Center Networks

Yuliang Li
Rui Miao       Changhoon Kim       Minlan Yu

USC University of Southern California
BAREFOOT NETWORKS
FlowRadar:
Capture all flows on a fine time scale
Flow coverage

- We need to inspect each individual flow
  - Defined by 5 tuples: source, dest IPs, ports, protocol

Transient loops/blackholes

Fine-grained traffic analysis
Temporal coverage

• We need flow information on a fine time scale

Short time scale Loss rates

Timely attack detection
Key insight: division of labor

- Goal: Monitor all the flows on a fine time scale

Collector has limited bandwidth and storage

Overhead at the collector

Mirroring

Needs sampling

Overhead at the switches

NetFlow

Limited per-packet processing time

Limited memory (10s of MB)
Key insight: division of labor

- Goal: Monitor all the flows on a fine time scale

- Use fixed operations per packet at switches

- Overhead at the collector

- Mirroring

- Small data structures

- NetFlow

- Overhead at the switches

FlowRadar
FlowRadar architecture

Collector

Extract per-flow counters across switches

Analysis Applications

Flows & counters

Switches

Compact flow counters with fixed per-packet operations

Periodic report
Challenge: How to handle collisions?

- Handling hash collisions is hard
  - Large hash tables $\rightarrow$ high memory usage
  - Linked list/Cuckoo hashing $\rightarrow$ multiple, non-constant #memory accesses
Solution: Embraces collisions

- **Handling** hash collisions is hard
  - Large hash tables $\rightarrow$ high memory usage
  - Linked list/Cuckoo hashing $\rightarrow$ multiple, non-constant #memory accesses

- **Embracing** the collisions
  - Less memory and constant #memory accesses
Solution: Embraces collisions

- Embracing the collisions
- Less memory and constant #memory accesses

Counting table

<table>
<thead>
<tr>
<th>FlowXor</th>
<th>a</th>
<th>a ⊕ b</th>
<th>b ⊕ c</th>
<th>b ⊕ c</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlowCount</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PacketCount</td>
<td>S(a)</td>
<td>S(a)+S(b)</td>
<td>S(b)+S(c)</td>
<td>S(b)+S(c)</td>
<td>S(a)</td>
<td>S(c)</td>
</tr>
</tbody>
</table>

H1

H2

H3

S(x): #packets in x

[Invertible Bloom Lookup Table (arXiv 2011)]
Flow Filter to identify new flows

1. Check and update the flow filter
2. Update counting table

- The first packet from a new flow, update all fields
- Subsequent packets update only PacketCount

<table>
<thead>
<tr>
<th>FlowXor</th>
<th>a</th>
<th>a⊕b</th>
<th>b⊕c⊕d</th>
<th>b⊕c⊕d</th>
<th>a</th>
<th>c⊕d</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlowCount</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PacketCount</td>
<td>S(a)</td>
<td>S(a)+S(b)</td>
<td>S(b)+S(c)</td>
<td>S(b)+S(c)</td>
<td>S(a)</td>
<td>S(c)</td>
</tr>
</tbody>
</table>

PacketCount updates:
+1 +1 +1 +1

PacketCount:
S(a) S(b) S(c) S(a) S(b) S(c)

Enclosed Flowset

[Invertible Bloom Lookup Table (arXiv 2011)]
Easy to implement in merchant silicon

• **Switch data plane**
  – Fixed operations per packet
  – Small memory: 2.36MB for 100K flows

• **Switch control plane**
  – Collects the small encoded flowset every 10ms

• **We implemented it using P4 Language.**
  – Portable to many p4-compatible switch chips
Flows & counters

Extract per flow counters across switches

Analysis Applications

FlowRadar architecture

Compact flow counters with fixed per packet operations

Collector

Stage1. Single Decode

Stage2. Network-wide Decode

Periodic report

Switches

Encoded flowset

Encoded flowset

Encoded flowset
Stage 1. SingleDecode

Input:
an encoded flowset from a single switch

Output:
per-flow counters

Flow filter

Bloom filter

FlowXor  …
FlowCount  …
PacketCount  …

Counting table

<table>
<thead>
<tr>
<th>Flow</th>
<th>#packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>S(a)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Stage 1. SingleDecode

- Find a **pure cell**: a cell with one flow
- Remove the flow from all cells

<table>
<thead>
<tr>
<th>FlowXor</th>
<th>FlowCount</th>
<th>PacketCount</th>
<th>Flow</th>
<th>#packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>5</td>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>a⊕b</td>
<td>2</td>
<td>12</td>
<td>b⊕c⊕d</td>
<td>13</td>
</tr>
<tr>
<td>b⊕c⊕d</td>
<td>3</td>
<td>13</td>
<td>c⊕d</td>
<td>6</td>
</tr>
</tbody>
</table>

Decoded:

Flow filter

Pure cell
Stage 1. Single Decode

- Find a **pure cell**: a cell with one flow
- Remove the flow from all cells
  - Create more pure cells
- **Iterate until no pure cells**

<table>
<thead>
<tr>
<th>FlowXor</th>
<th>Flow</th>
<th>#packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>b\oplus c \oplus d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
</tbody>
</table>

Decoded: a 5

Flow n Packet n
a 5

Flow filter

\[ \begin{array}{ccc}
H1 & H2 & H3 \\
\end{array} \]
**Stage 1. Single Decode**

Decoded:

<table>
<thead>
<tr>
<th>Flow</th>
<th>#packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
</tbody>
</table>

We want to leverage the network-wide info to decode more flows.

<table>
<thead>
<tr>
<th></th>
<th>FlowXor</th>
<th>FlowCount</th>
<th>PacketCount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c ⊕ d</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c ⊕ d</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
FlowRadar architecture

Collector

Stage1. Single Decode

Stage2. Network-wide Decode

Analysis Applications

Periodic report

Encoded flowset

Encoded flowset

Encoded flowset
Key insight: overlapping sets of flows

- The sets of flows overlap across hops
  - We can use the redundancy to decode more flows
- Use different hash functions across hops

<table>
<thead>
<tr>
<th>FlowXor</th>
<th>FlowXor</th>
<th>FlowCount</th>
<th>FlowCount</th>
<th>PktCount</th>
<th>PktCount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>a⊕c</td>
<td>b⊕c</td>
<td>a⊕b</td>
<td>b⊕d</td>
</tr>
<tr>
<td>FlowCount</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>PktCount</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>FlowXor</td>
<td>a⊕d</td>
<td>a⊕c</td>
<td>b⊕c</td>
<td>a⊕b</td>
<td>b⊕d</td>
</tr>
<tr>
<td>FlowCount</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Use 5 cells to decode 4 flows

Collector can leverage flow sets from all switches to decode more flows
Key insight: overlapping sets of flows

- The sets of flows overlap across hops
  - We can use the redundancy to decode more flows
- Use different hash functions across hops
- Provision switch memory based on $\text{avg}(\#\text{flows})$, not $\text{max}(\#\text{flows})$
  - SingleDecode for normal cases
  - Network-wide decode for bursts of flows
Challenge 1: sets of flows not fully overlapped

- Flows from one switch may go to different next hops
- One switch receives flows from multiple hops
Solution: Zigzag decode with Flow Filters

- Generalize to the entire network
  - No need for routing information
  - Support incremental deployment

Decoded:

```
| a | b | c | d |
```

FlowDecode

```
| a | c | d | e |
```

SingleDecode

Remove from the other

SingleDecode

Flow filter

Flow filter
Challenge 2: different counter values

- The counters of the same flow may be different across hops
  - Some packets may get lost
  - Some packets may be on the fly
Solution: calculate counters for each switch

• Solve a linear equation system for each switch
  – Already know the full set of flows at each switch
  – Solvable with n flows and >= n combinations of counters

\[
\begin{align*}
S(a) + S(b) + S(d) &= 14 \\
&\ldots
\end{align*}
\]
Solving the linear equation system

• Challenge: solving the linear equation system is not fast
• The matrix is very sparse, each column has k “1”s.
• We use iterative solvers
• Speed up the iteration:
  – Start the iteration from close approximations of counters, which is got from the FlowDecode
  – Stop the iteration when the result is floating within ±0.5 an integer.
# FlowRadar architecture

## Collector

### Stage1.SingleDecode

- Flow filter
- FlowXor
- FlowCount
- PacketCount

### Stage2.1 FlowDecode

<table>
<thead>
<tr>
<th>Flow</th>
<th>#pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

### Stage2.2 CounterDecode

<table>
<thead>
<tr>
<th>Flow</th>
<th>#pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
</tbody>
</table>

## Switches

### FlowXor

### FlowCount

### PacketCount

## Flows & counters

Analysis Applications

### FlowXor

### FlowCount

### PacketCount

<table>
<thead>
<tr>
<th>Flow</th>
<th>#pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>z</td>
<td>6</td>
</tr>
</tbody>
</table>
Evaluations

- Memory usage is efficient
- Bandwidth usage is low
- NetDecode vs SingleDecode
Evaluation

- Simulation of k=8 FatTree (80 switches, 128 hosts) in ns3
- Configure the memory base on $\text{avg}(\#\text{flow})$,  
  - when burst of flows happens, use network-wide decode
- The worst case is all switches are pushed to $\text{max}(\#\text{flow})$  
  - Traffic: each switch has same number of flows, and thus same memory
- Each switch reports the flowsets every 10 ms.
Memory efficiency

FlowRadar: 24.8MB
FlowRadar: 2.36MB

#cell=#flow (Impractical)
Bandwidth usage

- Evaluated based on Facebook data centers (SIGCOMM’15)
  - Each ToR has less than 100K flows per 10ms → less than 2.3Gbps
  - Each ToR talks to 44 * 10G-hosts
  - FlowRadar consumes 0.52% of total ToR bandwidth
NetDecode vs. SingleDecode

NetDecode needs more time

SingleDecode: 100K flow, 10ms

NetDecode: 126.8K flow, 3sec
NetDecode

• The CounterDecode takes the majority of the decoding time
• The CounterDecode limits the #counters could be decoded
  – If #flows > 126.8K, we can still decode all flows (up to 152K) without counters using FlowDecode
FlowRadar analyzer

Collector

Stage1. Single Decode

Stage2. Network-wide Decode

Switches

Encoded flowset

Encoded flowset

Encoded flowset

Analysis Applications

Flows & counters

Periodic report
Analysis applications

• Flow coverage
  – Transient loops/blackholes
  – Error in match-action table
  – Fine-grained traffic analysis

• Temporal coverage
  – Short time scale per-flow loss rate
  – ECMP load imbalance
  – Timely attack detection
Per-flow loss map: better temporal coverage

- Detect losses after every flowlet
FlowRadar conclusion

• Report all per-flow counters on a fine time scale
• Fully leverage both switches and the collector
  – Switches: Encoded flowsets with fixed per-packet processing time, and low memory usage
  – Collector: Network-wide decoding
LossRadar:
Detect individual losses on a fine time scale
Losses have significant impact

• Significant latency increase and throughput drop
  – Violating SLA and drop revenue

• Takes operators up to tens of hours to find and fix the problem

Takeaway:
We want to know loss ASAP
Challenge: many types of losses

Takeaway:
We need a generic tool
Challenge: losses could happen everywhere

Takeaway: We need the locations of the losses

TCP: I have losses
But I don’t know where
Challenge: lack of details of losses

- Difficult to infer the root causes

Takeaway:

Need the information of individual losses

different loss patterns over time

Different flows experience different loss patterns over time
LossRadar overview

- Fast detection
- Knowing location
- Need info of individual losses
- Being generic
How to cover the whole network

- Need to cover all pipelines

UM: Upstream meter
DM: Downstream meter

Cover ingress pipeline

Cover egress pipeline
LossRadar overview

• Fast detection → Periodically send traffic digest to collector
• Knowing location → Install meters to monitor traffic
• Need info of individual losses → Include details of each loss
• Being generic → Compare the sets of packets
How to compare the sets of packets

• Using Invertible Bloom Filter (IBF) [Sigcomm’ 11]
• Only $O(#\text{loss})$ memory
  – Each end keeps a small piece of memory
  – Same packets at both ends will cancel out
  – Only the differences remain
How to achieve low memory usage

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

upstream

downstream

Collector
How to achieve low memory usage

a = 5-tuple + IPID

<table>
<thead>
<tr>
<th>xorSum</th>
<th>a</th>
<th>0</th>
<th>a</th>
<th>a</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xorSum</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

upstream

download

drop

Collector
How to achieve low memory usage

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>a⊕b</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

upstream

downstream

Collector
How to achieve low memory usage

**xorSum count**

<table>
<thead>
<tr>
<th></th>
<th>a⊕c</th>
<th>b⊕c</th>
<th>a⊕b</th>
<th>a⊕c</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**xorSum count**

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

upstream

drop

downstream

Collector
How to achieve low memory usage

### Collector does subtraction

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a⊕c⊕d</td>
<td>3</td>
</tr>
<tr>
<td>b⊕c</td>
<td>2</td>
</tr>
<tr>
<td>a⊕b⊕d</td>
<td>3</td>
</tr>
<tr>
<td>a⊕c</td>
<td>2</td>
</tr>
<tr>
<td>b⊕d</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>b⊕d</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b⊕d</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xorSum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a⊕c</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a⊕c</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Collector does subtraction.

**Collectors**

- **upstream**
- **downstream**

Collector
Benefits

• Memory-efficient
  – Proportional to #losses
  – 10KB per meter (traffic pattern reported in DCTCP)

• Extend to collect more packet information
  – TTL: help identify loops
  – Timestamp: tagged in the packet headers at the upstream
  – Any other fields that programmable switches can configure
Challenges

• How to ensure UM and DM capture the same set of packets
  – Use the packet header to carry batch ID

• Incremental deployment
  – Put UM and DM around the blackbox
  – Compare sum(UM_i) and sum(DM_j)
LossRadar conclusion

• Design a fast and efficient loss detection system
  – Provide locations and details of individual losses
  – Generic to any types of losses
  – Memory only proportional to #losses

• Future work: design an analyzer to diagnose problems
  – Temporal analysis, correlation across flows and switches
  – Combine with info provided by hosts
Conclusion

• **Full visibility of Data Centers**
  – Report all the flows and all the lost packets in a few milliseconds across all the switches

• **Efficient data structures on programmable switches**
  – Both FlowRadar and LossRadar are implemented on P4
  – Technology transfer to Barefoot switches (See my demo tomorrow)