HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows

Junzhi Gong¹, Tong Yang¹, Haowei Zhang¹, Hao Li¹, Steve Uhlig², Shigang Chen³, Lorna Uden⁴, Xiaoming Li¹

Peking University¹, Queen Mary University of London², University of Florida³, Staffordshire University⁴
Finding Top-k Elephant Flows

Top-k Elephant Flows

Basic Flow Statistics
Finding Top-k Elephant Flows

Finding top-k elephant flows serves as a fundamental network management function.

Applications

- Congestion Control
- Anomaly Detection
- Network Capacity Planning

Top-k Elephant Flows
Flow Distribution of Real Traffic

- Highly Skewed
  - The majority are mouse flows
  - The minority are elephant flows
  - However, elephant flows are much more important
Challenges

- **High Line Rate**
  - Impossible to track information of all flows
  - Solution: approximate methods

- **High Latency of Off-chip Memory**
  - Force algorithm to use on-chip memory, like SRAM
  - The size of on-chip memory is small, e.g., several megabytes
Existing Solution I: Count-All

- A sketch to approximately record all flow sizes
- And a min-heap to maintain top-k elephant flows
- Sketches are smaller than hash tables
  - But they still need to store all flows, which is not memory efficient
Existing Solution II: Admit-All-Count-Some

- Frequent, Lossy Counting, Space-Saving, CSS
- Process all flows, but only record a small part of them
- Example: Space-Saving

\[ f_1, 15 \quad f_2, 10 \quad f_3, 7 \quad f_5, 6 \]
Limitation of two existing solutions

- **Count-All**
  - Spend too much memory and time on mouse flows
  - With total memory size small, the accuracy cannot be high

- **Admit-All-Count-Some**
  - Could drastically over-estimate flow sizes (Space-Saving, CSS)
  - Could make the size of elephant flows inaccurate (Frequent, Lossy Counting)
  - With limited memory size, many mouse flows can be mistreated as elephant flows.
Our Contributions

- A new data structure, named HeavyKeeper, which achieves high accuracy and high processing speed in finding top-k elephant flows.
- Experiments on real network traffic and synthetic datasets, showing the high performance of HeavyKeeper
- Deploy algorithms in Open vSwitch (OVS) platform
Our Solution

- **HeavyKeeper**
  - Strategy: count-with-exponential-decay
  - Keeps only elephant flows
  - Drastically reduce space wasted on mouse flows

- **Exponential-weakening-decay**
  - Mouse flows are easily be decayed and removed
  - Elephant flows can hardly be removed
  - Will not admit new flows unless one flow is ready to be removed
Data Structure

$w$ buckets

$F_3$

$h_1(f_3)$

$h_2(f_3)$

$h_d(f_3)$

$F_3 7$ $F_4 3$ $F_1 22$ $\cdots$ $F_5 1$

$F_1 22$ $F_2 14$ $F_5 1$ $\cdots$ $F_4 2$

$F_4 3$ $F_1 21$ $F_2 14$ $\cdots$ $F_5 1$

$d$ arrays

FP: fingerprint field  C: counter field
Insertion

Map: each incoming flow is mapped to one bucket for each array by using the hash functions

Insert: for each mapped bucket, different strategies are applied according to different cases.
Insertion

Case 1: if C=0
then C=C+1

Case 2: if C>0 && FP=F³
then C=C+1

Case 3.1: if C>0 && FP ≠ F³
with prob. = 1 - \( P_{reduce} \), C=C

Case 3.2: if C>0 && FP ≠ F³
with prob. = \( P_{reduce} \), C=C-1
Case 1

- The bucket is empty, i.e., $C=0$
  - Simply set the fingerprint to $\text{FP}(f_3)$, and set $C=C+1=1$
Case 2

- The bucket is not empty, i.e., $C > 0$, and $FP = F_3$
  - Simply set $C = C + 1$
Case 3

- The bucket is not empty, i.e., $C>0$, but $FP \neq F_3$
  - With a probability $P_{reduce}$, decrease the counter $C$ by 1.
  - If the counter is reduced to 0, we replace the original FP to the fingerprint of the new flow $F_3$
Query

- For incoming flow $f$
- Map: get $d$ mapped buckets
- Filter: get those buckets whose fingerprint is $FP(f)$
- Answer: report the largest value among all filtered buckets (0 if no bucket left after filtering)
Decay Probability

- \( P = b^{-c} \)
- \( b \) is a predefined constant, e.g., \( b = 1.08 \)
- \( c \) is the value in the current counter field (the value to be decayed)
- The larger \( c \) is, the harder its flow size is decayed
When the original bucket stores a mouse flow, it will be easily decayed.

Decay with probability $1.08^{-3} = 0.794$
Decay Probability

- When the size of mouse flow is decayed to 0, the original flow will be replaced
  - Therefore, mouse flows can hardly stay in HeavyKeeper

\[
D = 1.08^{-1} = 0.926
\]
Decay Probability

- When the original bucket stores an elephant flow, it can be hardly decayed
  - Therefore, elephant flows can be stably stored in HeavyKeeper
  - The estimated size of elephant flows will also be accurate

\[
\text{Decay with probability } 1.08^{-100} = 4.55 \times 10^{-4}
\]
Analysis

- With exponential-weakening decay, HeavyKeeper will tend to store elephant flows and evict mouse flows.
- Most mouse flows are simply *passers-by* of HeavyKeeper.
- Elephant flows are easily stored, and their estimated flow sizes are also accurate.
Basic Algorithm

- To find top-k elephant flows, the basic version of our algorithm will simply use a min-heap to maintain the top-k elephant flows, like sketches.
Analysis

 Complexity

- Space: $O(d^w)$.
  - Experiments will show HeavyKeeper achieves high accuracy with very small memory usage.
- Time: $O(k)$.
  - Updating the min-heap is time consuming. Even with the help of hash tables, the insertion of the min-heap will also consume $O(\log k)$ time complexity.
Optimizations

- Using the min-heap, improving accuracy
  - Too many details, skip

- Replacing the min-heap with a single list, improving speed
  - Ignoring fingerprint collisions, the flow size of each flow will grow 1 by 1.
  - Define a threshold $T$ (e.g., $T=1000$)
  - Recording the incoming flow in the list if the estimated size of the flow is equal to $T$
  - Time complexity $O(d)$

\[ f_j \xrightarrow{} \text{HeavyKeeper} \xrightarrow{} \hat{f}_j \xrightarrow{} \hat{f}_j = T? \xrightarrow{Y} \text{List} \]
Experimental Setup

- **Datasets**
  - Campus network traffic
  - CAIDA
  - Synthetic skewed datasets, Zipf

- **Implementation**
  - $d = 2$, enough for HeavyKeeper
  - Fixing $k$ and changing memory size, estimate accuracy
  - Fixing memory size and changing $k$, estimate accuracy
  - Speed evaluation
Experimental Setup

Metrics

- Average absolute error (AAE). Absolute error is defined as $|n_j - \hat{n}_j|$.

- Average relative error (ARE). Relative error is defined as $\frac{|n_j - \hat{n}_j|}{n_j}$.

- Precision. Among the $k$ reported flows, how many flows are real top-$k$ elephant flows. 0%~100%.
Changing Memory Size

- **log_{10}AAE**
- **log_{10}ARE**
- **Precision**
Changing K

![Graph 1: log10(AAE) vs. k](image1)

![Graph 2: log10(ARE) vs. k](image2)

![Graph 3: Precision vs. k](image3)
Speed Evaluation

![Histogram showing throughput (Mps) vs memory size (KB) for SS, CM Sketch, LC, and HeavyKeeper. The x-axis represents memory size in KB, ranging from 1 to 5, and the y-axis represents throughput in Mps, ranging from 0 to 25. Each bar color corresponds to a different method: red for SS, purple for CM Sketch, blue for LC, and white for HeavyKeeper. The throughput values for each method at each memory size are visually compared to determine which performs the best.]
Open vSwitch Deployment

- Create a shared buffer between processes to store flow IDs.
- Modify OVS datapath to report each incoming flow ID to the shared buffer.
- A user-space program of HeavyKeeper to process those flow IDs from the buffer.
Evaluation

![Graph showing throughput comparison between different algorithms: OVS, HeavyKeeper, CM Sketch Algorithm, SS, and LC. The graph indicates that OVS and HeavyKeeper have higher throughput compared to the others.]
Thank you!