SAROS: A Self-Adaptive Routing Oblivious Sampling Method for Network-wide Heavy Hitter Detection

Enhan Li[1,2], Wenhao Wu[2], Zhaohua Wang[3], Zhenyu Li[2], Jianwei Niu[1,4]
[1]Hangzhou Innovation Institute, Beihang University
[2]Institute of Computing Technology, Chinese Academy of Science
[3]Computer Network Information Center, Chinese Academy of Sciences
[4]Zhongguancun Laboratory

ABSTRACT

Network-wide heavy hitter detection is usually performed by sampling on several network measurement points (NMPs) and merging the measurement results in the centralized controller to get a network-wide view. However, a packet may pass several NMPs and be counted multiple times when measurement results are merged, which causes the double-counting problem and leads to incorrect detection. Existing studies either overlook this problem or require significant memory usage. This paper proposes **SAROS**, a self-adaptive routing oblivious sampling method for accurate network-wide heavy hitter detection. Specifically, SAROS exploits a sampling mechanism in the data plane, where the sampling threshold on each measurement point is predicted and adaptively set by the control plane. Such guidance from the control plane greatly reduces the memory usage in the data plane, while mitigating the double-counting problem. Experimental results show that, compared with existing solutions, SAROS improves the F1-Score of heavy hitter detection by 10~40%.

CCS CONCEPTS

• **Networks → Network measurement**; *Programmable networks*; Network monitoring; Denial-of-service attacks.

KEYWORDS

Network-Wide Heavy Hitter Detection, Self-Adaptive Sampling, Programmable Switch

ACM Reference Format:
Enhan Li, Wenhao Wu, Zhaohua Wang, Zhenyu Li, Jianwei Niu. 2024. SAROS: A Self-Adaptive Routing Oblivious Sampling Method for Network-wide Heavy Hitter Detection. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 3–4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663429

1 INTRODUCTION

Network-wide heavy hitter (*i.e.* flows with unexpectedly high volume) detection is essential for various network management tasks like DDoS detection [3, 23, 27], intrusion detection [22] and load banlancing [15]. Current network-wide heavy hitter detection methods usually involve an architecture with multiple network measurement points (NMPs) and a central controller. NMPs use packet sampling or maintain an abstract of passed flow with special data structures, while the central controller collects all these sampled packets or data structures and merges them to create a network-wide view for heavy hitter detection.

In this context, programmable switch [9] has emerged as a promising approach for NMPs implementation with better performance and flexibility. Programmable switches maintain compact data structures (*e.g.* CM-Sketch [10], UnivMon [20]) to record traffic information in each NMP. However, simply taking sum or maximum [25] to merge the data structures maintained on NMPs cannot obtain an accurate network-wide view. When a packet traverses several NMPs, it will be recorded multiple times by different NMPs and calculated accumulatively in network-wide view due to the lack of topology and routing information. Such phenomenon results in the *double-counting* problem and incorrect measurement results. Therefore, achieving network-wide heavy hitter detection with routing-oblivious measurement approach is of critical importance.

Unfortunately, existing sketch-based network measurement systems (*e.g.* Sonata [17], Poseidon [26], Jaqen [21]) overlook the double-counting problem in scenarios of network-wide measurement. Some network-wide heavy hitter detection methods (HHD [18], NWHHD [11]) also have strong constraints of network topology. These approaches cannot work in the situation out of the assumptions like node failures or routing changes. Recently, several studies have focused on routing-oblivious measurement. INVEST [12] utilizes the cardinality estimation property of Hyperloglog [14] to eliminate double counting but does not retain flow ID-related information for heavy hitter detection. AROMA [5] addresses the issue of duplicate packets relying on a k-partition hash-based structure. However, AROMA requires significant memory usage. AROMA needs 16.2MB in IPv4 scenarios and 39.1MB in IPv6 scenarios[1], while a commercial programmable switch (*e.g.* Intel-Tofino) has only 20MB of on-chip storage.

In this paper, we propose **SAROS** (Self-Adaptive Routing Oblivious Sampling) for network-wide heavy hitter detection. SAROS designs a sampling mechanism in the data plane with almost negligible memory consumption. The control plane merges all sampled packets, retaining and using their statistical data for traffic estimation and heavy hitter detection. The sampling threshold on each measurement point is predicted adaptively by the control plane.

Corresponding author: Zhenyu Li.

[1]Set $k = 10^6$ in k-partition hash-based structure

Such guidance from the control plane greatly reduces the memory usage in the data plane, while mitigating the double-counting problem.

To sum up, we make the following contributions:

- We propose a double-counting-free sampling mechanism for network-wide heavy-hitter detection, which ensures detection accuracy in routing oblivious cases.
- We propose a dynamic sampling threshold prediction method in the SAROS control plane, which calculates appropriate sampling thresholds for each NMP according to network traffic distribution.
- We implement a SAROS prototype in programmable switches and make it publicly available [1]. Compared with existing solutions, SAROS has improved the heavy hitter detection accuracy by 10~40%, and reduced memory consumption by 99%.

2 PRELIMINARIES

2.1 Problem Definitions

The network is composed of n network measurement points (switches) $\mathcal{N}$. The entire network is abstracted as one big switch. We assume that each packet passes through at least one network measurement point.

Each packet is represented as $< x, i >$, where $x \in U$ is a flow identifier like a five-tuple (source IP, destination IP, source port, destination port, protocol), a source or destination IP pair, etc. and i is a unique packet identifier. For example, we can regard the sequence numbers in TCP protocols as packet identifiers. Previous works [13, 28] suggest various methods to identify packets according to their header fields.

A flow is a stream of packets with the same flow identifier $x \in U$. We use f_x representing the total number of data packets in the flow with the identifier x.

The actual network-wide volume is denoted as $|S|$. The task of identifying network-wide heavy hitters is defined as follows: At regular time intervals, a flow x is considered a heavy hitter if

$$x/|S| \geq \theta(0 < \theta < 1)$$

. The actual set of network-wide heavy hitters is denoted as $H' \in U$. The system returns a set of network-wide heavy hitters $H \in U$.

2.2 K minimum value algorithm

K minimum value algorithm (KMV) [4] is based on the idea of hash distance estimation, using the properties of random hashing to approximate the number of distinct elements in a data stream. The intuition behind KMV is as follows: assuming the existence of a very uniform hash function that uniformly maps the set of elements $S = \{x_1, x_2, ...x_n\}$ to the interval [0,1]. Because the hash function is uniform, the average distance between hashed values can be used to estimate the number of distinct elements in S.

It is composed of three operations: ADD, QUERY, and UNION. The ADD operation hashes incoming elements uniformly into the [0,1] interval. The QUERY operation retrieves the top k smallest hash values as the KMV set and estimates the number of elements using the maximum value in the KMV set as the query output. The UNION operation combines the KMV set of multiple KMV instances

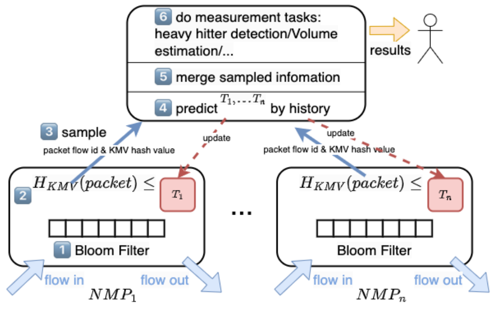
6 do measurement tasks: heavy hitter detection/Volume estimation/...
5 merge sampled infomation
4 predict $T_1, ...T_n$ by history
results
ControlPlane
3 sample
packet flow id & KMV hash value
update
update
packet flow id & KMV hash value
2
$H_{KMV}(packet) \leq$
T_1
$H_{KMV}(packet) \leq$
T_n
...
1 Bloom Filter
Bloom Filter
DataPlane
flow in
flow out
NMP_1
flow in
flow out
NMP_n

Figure 1: System architecture of SAROS.

into one big KMV set which contains the top k smallest hash values in all instances. The UNION operation merges two instances A and B into a unified instance for monitoring $S_A \bigcup S_B$, where S_A and S_B are the streams processed by A and B, respectively. Let k be the minimum size of instances A and B, and select k smallest hash values from the union of the two instances $A \bigcup B$ as the return value of the UNION operation. Similarly, the KMV algorithm can also merge multiple instances by performing multiple UNION operations.

However, it is challenging to directly deploy the KMV algorithm in the programmable data plane because the real-time selection of k data packets with the smallest hash values among all packets passing through the NMPs poses a certain difficulty. Therefore, we further explore how to deploy the KMV algorithm in the system.

3 SYSTEM DESIGN

As shown in Figure 1, SAROS consists of two main parts: the data plane deployed at all NMPs and the control plane deployed on the central controller. The KMV-based algorithm is implemented collaboratively by the data plane and the control plane. Specifically, we maintain a dynamic threshold T in each data plane to sample the packets with hash values less than T. The control plane merges all sampled packets, retaining the top k packets with the smallest hash values, and uses a KMV-based method for heavy hitter detection or other measurement tasks. Furthermore, the control plane predicts the dynamic sampling threshold T based on historical sampling data and disseminates it to each NMP for updates.

3.1 Data Plane Sampling Module

The intuitive solution for the data plane in routing-oblivious measurement is treating each NMP as a KMV instance and maintaining a min-heap for the top k packets with the smallest hash values. However, it is hard for the data plane to maintain a min-heap due to storage and computation constraints. Instead, we only maintain a dynamic threshold T in each NMP and consider that the packets having smaller hash values than T are in the 'min-heap'. Then we send all these packets to the central controller to get the global k-minimum packets. The dynamic threshold T will be adaptively updated to avoid network congestion caused by over-sampling and estimation errors caused by under-sampling.

The SAROS data plane design is detailed in Algorithm 1. We set a dynamic sampling threshold at each NMP and update it by the central controller at every time interval, which can guarantee that

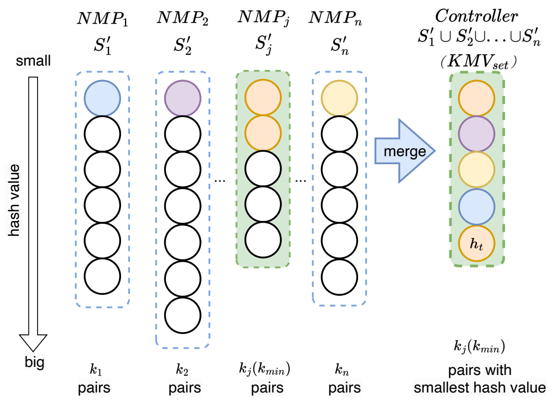

Figure 2: Sampling process in the data plane of SAROS.

Algorithm 1: SAROS data plane algorithm.

```
Input: Flow $S_j$ passing through $NMP_j$; dynamic sampling
       threshold $T_j$ at the measurement point, initialized to
       0.3; KMV parameter set to $k$; hash function $H_{KMV}$;
       $N_h$ hash functions, Bloom Filter $B$ of length $N_b$; time
       interval $t$.
Output: Sampled packets.
1  Function RoutingObliviousSample(input parameters):
2      Initialize KMV set with the first $k$ values;
3      while in time interval t do
4          Perform $N_h$ different hashes on the flow ID $x$ of the
             packet;
5          if the $N_h$ corresponding positions in B are all 0 then
6              Use $H_{KMV}$ function calculate KMV hash $h_{(x,i)}$:
                 $h_{(x,i)} \leftarrow H_{KMV}(x, i)$;
7          if $h_{(x,i)} <= T_j$ then
8              Sample $(x, h_{(x,i)})$ to the control plane;
9          Set $N_h$ corresponding positions in $B$ to 1;
10     Reset Bloom Filter $B$;
```

approximately k packets with the smallest hash values are sampled. Our approach can achieve a balance between network-wide heavy hitter detection accuracy and data plane storage since it only needs to store a threshold on the data plane. The processing of each packet on the data plane is divided into the following two stpdf.

Step 1: Use a Bloom Filter to check if the packet has passed through the same measurement point already as shown in line 4 to line 5 in Algorithm 1. In the data plane, each measurement point has a Bloom Filter, denoted as B to verify whether a data packet is duplicated, reducing communication overhead between data and control planes. Initially, the packet's flow identifier x undergoes N_h hash functions, mapping to 0 to N_b. Then check the corresponding position in Bloom Filter B. If all values in the Bloom Filter are 1, the packet is treated as duplicate, needing no further processing. Otherwise, the packet will be processed by Step 2.

Algorithm 2: SAROS control plane algorithm.

```
Input: $k_1, k_2, ..., k_n$ packets of information sampled by $N$
       measurement points; time interval $t$; global flow
       threshold $\theta$.
Output: global flow set $H$; dynamic sampling threshold for
        each NMP in the next time interval.
1  Function RoutingObliviousMerge(input parameters):
2      for $NMP_j \in N$ do
3          $T_{(j,t)} = k \cdot \frac{h_j}{k_j}$;
4          Predict the dynamic sampling threshold $T_{(j,t+1)}$ for
             $NMP_j$ in the $t+1$ time interval based on the
             sequence $T_{(j,t)}$;
5          Send $T_{(j,t+1)}$ to $NMP_j$ as dynamic threshold in next
             time interval;
6      $k_{min} \leftarrow Min(k_1, k_2, ..., k_N)$;
7      $KMV_{set} \leftarrow [(x_1, h_1), (x_2, h_2), ..., (x_{k_{min}}, h_{k_{min}})]$;
8      $h_t$ is set by the maximum hash value in $KMV_{set}$;
9      if exist a flow $x \in KMV_{set}$ satisfies $\frac{f_x}{k_{min}} \geq \theta$ then
10         Flow $x$ is added to the heavy hitter set $H$;
11     Output the heavy hitter set $H$;
```

Step 2: We use a dynamic threshold to implement uniform sampling in the KMV algorithm, which is detailed in line 5 to line 8 in Algorithm 1. As shown in Figure 2, we use the hash function H_{KMV} which is shared across all NMPs to hash the flow identifier x and packet identifier i of the data packet (x, i) to obtain the KMV hash value $h_{(x,i)}$. For example, TCP data packets use the five-tuple as the flow identifier and the TCP sequence number as the packet identifier. If the hash value $h_{(x,i)}$ is smaller than the dynamic sampling threshold T_j at NMP_j, it is considered as one of the top k smallest hash values in this NMP. Then the NMP sends the corresponding flow identifier and hash value pair $(x, h_{(x,i)})$ to the central controller to facilitate various measurement tasks, especially those requiring flow ID information. At the end of the time interval, each NMP may sample about k flow identifier and KMV hash value pairs to the central controller with the smallest KMV hash values.

Therefore, the data plane approximately achieves sampling the first k packets with the smallest hash values in the data plane NMP_j only using a dynamic threshold T_j.

3.2 Control Plane Estimating Algorithm

After receiving the sampling flow identifier and KMV hash value pairs from each measurement point, the control plane first predicts the size of the k-th minimum hash value in the next time interval at each measurement point based on historical data and sends this as a dynamic sampling threshold to each measurement point. Then, it utilizes the merging operation of KMV to obtain a uniform network-wide sample (approximately k data pairs) and uses this data to detect network-wide heavy hitters, estimate network-wide volume, and perform other measurement tasks.

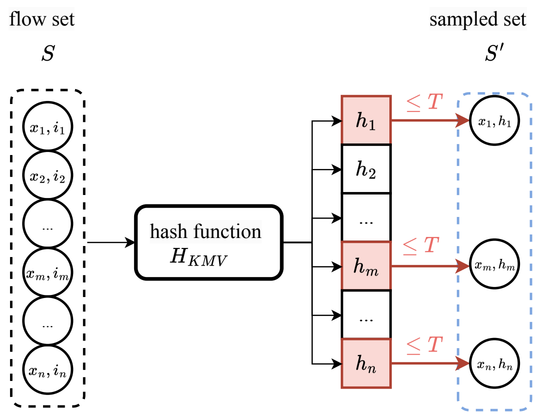

Figure 3: Merging process in the control plane of SAROS.

Predict dynamic sampling threshold for each NMP: The central controller updates the local dynamic sampling threshold for each measurement point in the data plane, as shown in lines 2 to 5 of Algorithm 2. The number of data packets that each measurement point should sample, as set by the administrator, is denoted as k. At the end of the t-th time interval, the central controller actually receives k_j sampled data packets from NMP_j, with the maximum hash value among these k_j packets being h_j. Since the hash H_{KMV} used for the KMV algorithm is uniform, based on the actual number of sampled data packets k_j and the maximum hash value h_j among these k_j packets at measurement point NMP_j, the average spacing of data packets passed NMP_j can be represented as (h_j/k_j). Therefore, the k-th smallest hash value $T_{(j,t)}$ at NMP_j in time interval t can be estimated as k times the average spacing h_j/k_j, i.e., $T_{(j,t)} = k \cdot \frac{h_j}{k_j}$.

Then based on the historical collection of k-th smallest hashes values $[T_{j,0}, T_{j,1}, ..., T_{j,t-1}, T_{j,t}]$, we use the EWMA (Exponentially Weighted Moving Average) algorithm to simply predict $T_{j,t+1}$ and push it to NMP_j updating the local dynamic sampling threshold for sampling in the time interval $t+1$. Thereby we form a self-adaptive sampling loop between the data plane and the control plane. Note that the prediction algorithm EWMA can also be replaced by more complex prediction models, such as LSTM [16], Transformer [24], etc.

Merge sampled information from all NMPs: The central controller merges flow identifier and KMV hash value in sampled packets from all NMPs for measurement tasks in the next. The parameter k set by the central controller for the KMV algorithm represents the expectation that each measurement point samples k data packets passing through that point with the smallest hash values within the time interval.

First, the central controller aggregates the information received from the NMPs. As shown in Figure 3, the sets of sampled data packets at n NMPs are denoted as S'_1, S'_2, ..., S'_n, with sizes of $k_1, k_2, ..., k_j, ..., k_n$ respectively at the end of time interval t. We denote k_{min} to be the minimum value among $k_1, k_2, ..., k_j, ..., k_n$.

Then we merge all the sets of packets and get the set of uniform sampling results $S'_1 \cup S'_2 \cup ... \cup S'_n$ with the size k_{min} based on the MERGE function of the KMV algorithm. As described in lines 6 to 8 of Algorithm 2, the central controller identifies the k_{min} data packets with top k_{min} smallest hash values among all sampled packets. Taking the whole network as one big switch, these k_{min} data packets are the smallest k_{min} hash values in the entire network. This approximately achieves uniform network-wide sampling and addresses the difficulty of sampling the same packet in different measurement points. Besides, we record the maximum hash value as h_t. Additionally, we can estimate total network traffic size and flow frequency based on merged information.

Network-wide heavy hitter detection: The central controller detects and outputs the network-wide heavy hitters using merged k_{min} packets information. Since the merged k_{min} packets information represents a uniform sampling result across the whole network, the distribution of k_{min} data packets can represent the distribution of network traffic across the entire network. Thus, if the proportion of the number of flow x in k_{min} packets is greater than θ, i.e., $\frac{f_x}{k_{min}} \geq \theta$, flow x is identified as a network-wide heavy hitter and will be added to the heavy hitter set H and alerted to the network administrator.

Furthermore, if the network administrator needs to reset the fixed sampling threshold θ or the number of sampled data packets k based on measurement requirements, there is no need to restart the data plane. It is sufficient to simply modify the control plane code and rerun the control plane.

4 EVALUATION

4.1 Experimental Setups

We have built the SAROS prototype and made it publicly available [1]. We implement data plane with P4 in the Bmv2 environment and a central controller with Python.

Network Topology. As shown in Figure 5, the experimental test network is configured with a Fat-Tree consisting of 4 pods, totaling 20 programmable switches, and 8 hosts. Each switch will randomly route the packet to a neighboring switch. We set the Time To Live (TTL) of each packet to 4 to avoid routing loops.

Datasets. We conduct simulation experiments with two dataset [6, 8]. We select two segments from MAWI dataset [8], labeled as MAWI1 and MAWI2. We also use the UNIV1 dataset [6] collected from a data center. Packets in each dataset will be hashed to the hosts in the experimental topology and enter the test network.

Baseline. We compare SAROS with the state-of-the-art network-wide heavy hitter detection method AROMA [5] in Bmv2 version. AROMA achieves network-wide routing-oblivious heavy hitter detection by maintaining a k-partition hash-based structure in the data plane and each partition can store the minimum hash value of the packet through the NMP. Just like SAROS, we also add a bloom filter to AROMA to drop duplicate packets passing through the same NMP.

Metrics. We use the following performance metrics:

1) F1-Score: It is denoted as $\frac{2 \times precision \times recall}{precision + recall}$, where precision is the correct fraction of reported heavy hitters and recall is the

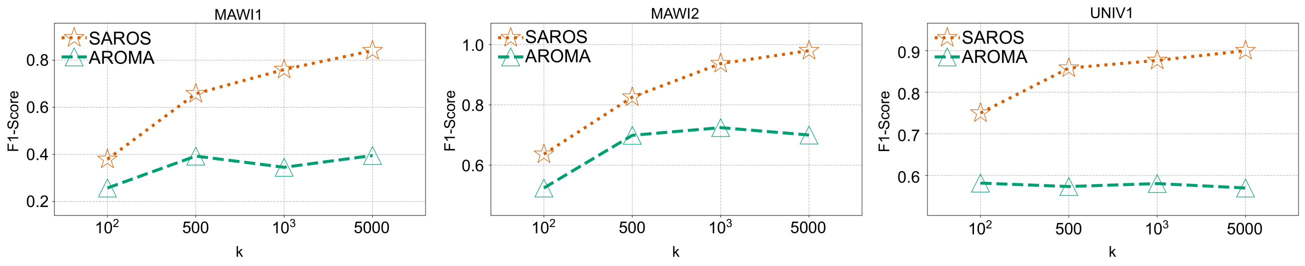

Figure 4: Under different k values, the comparison of F1-Score between SAROS and AROMA in three datasets.

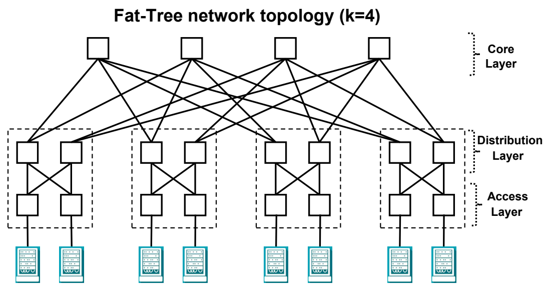

Figure 5: Network topology in experiments.

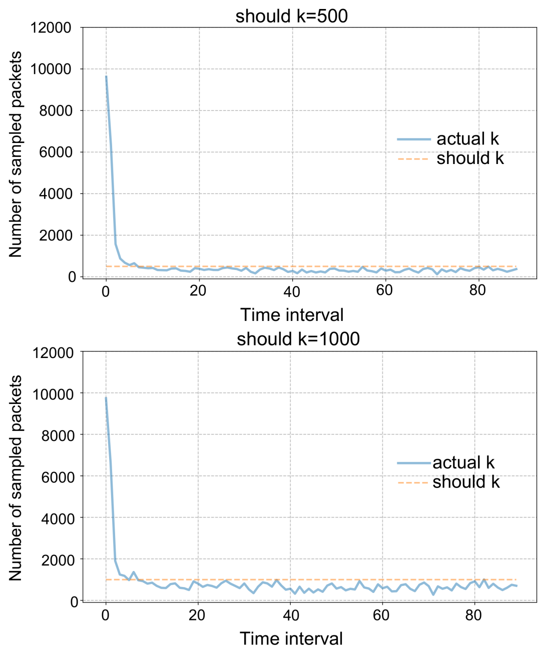

Figure 6: Convergence of SAROS under different k values (should k).

fraction of actual heavy hitters that are successfully reported. We use F1-Score to evaluate the performance of heavy hitter detection.

2) Communication volume: We evaluate it as the average number of messages sent from the data plane to the control plane per time interval.

3) Data plane memory usage: It measures the memory occupied by the data structures deployed in the data plane at each measurement point.

4.2 Heavy Hitter Detection

As shown in Figure 4, we compare the F1 scores across three datasets. We observe that as the number of sampled packets k increases, the F1 scores for both algorithms increase. Meanwhile, our SAROS consistently outperforms AROMA under the same k, especially under larger k.

The design of the SAROS data plane sampling method ensures higher reliability in heavy hitter detection. According to [7], in the hash-based estimation algorithm, the larger the number of packets counted, the smaller the error bound for estimating the total traffic volume. AROMA maintains k partition hash-based structure instances, each with the size of 1, while SAROS has 1 KMV instance on the data plane, with the size of k. We have a larger k value than AROMA, therefore SAROS have higher accuracy in heavy hitter detection.

Additionally, as shown in Figure 6, SAROS's adaptive sampling method demonstrates stable convergence, capable of converging within 10 time intervals, approaching sampling of approximately k data packets.

4.3 Communication Volume

Figure 7 illustrates the average communication messages sent from the data plane to the control plane per time interval. Overall, under the same KMV sample count k, the communication volumes of the two algorithms are comparable. Since AROMA samples k flow identifier and KMV hash value pair and SAROS samples about k flow identifier and KMV hash value pair, therefore, SAROS does not incur more communication overhead compared to AROMA.

4.4 Data Plane Memory Usage

SAROS requires a 32-bit dynamic threshold in each data plane, while AROMA needs to deploy a k-partition hash-based structure in which each partition stores a 32-bit hash value and a flow identifier with the size of 104 bits in the IPv4 scenario and 296 bits in the IPv6 scenario.

The on-chip memory of SAROS is about 24.4KB with a 32-bit register for the sampling threshold and a 24.4KB bloom filter. The

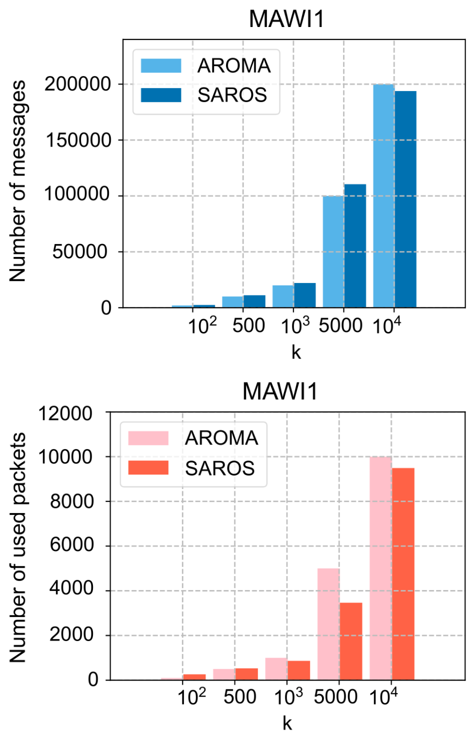

Figure 7: Under different k values, the comparison of communication volume between SAROS and AROMA in the MAWI1 dataset.

on-chip memory of AROMA in an IPv4 scenario can be calculated as $k(32bit + 104bit) + 24.4KB$, which includes k data structures consisting of 32-bit hash values and 104-bit flow IDs, and a $24.4KB$ bloom filter. In an IPv6 scenario AROMA memory consumption can be calculated as $k(32bit + 296bit) + 24.4KB$ with the same data structure as the IPv4 scenario, but needs 296 bits to record the flow ID. As shown in Table 1, the memory usage of AROMA in the data plane grows when parameter k increases and is up to 16.2 MB in the IPv4 scenario and 39.1 MB in the IPv6 or mixed scenario. The memory usage of SAROS is 24.4KB regardless of parameter k, much lower than the memory usage of AROMA.

Therefore, compared to the current state-of-the-art routing-oblivious network-wide heavy hitter detection methods, SAROS has extremely low memory occupation and is more suitable for deployment on programmable switches with limited on-chip memory.

5 RELATED WORKS

Network-wide heavy hitter detection. Existing network-wide heavy hitter detection methods make strong assumptions about topology and NMP deployments. NWHHD [11] makes assumptions that all the measurement points are sequentially connected; UnivMon-based [20] heavy hitter detection requires star-shaped topology with no duplicate packets in measurement points; Flow Radar [19] assumes that all flows only pass through a single and fixed path.

Table 1: In the IPv4 and IPv6 scenarios, the memory usage of data plane in AROMA and our SAROS under different k values.

Scenarios	Methods	k=10^2	k=10^4	k=10^6
IPV4	AROMA	26.1KB	190.4KB	16.2MB
	SAROS	24.4KB	24.4KB	24.4KB
IPV6	AROMA	28.4KB	424.4KB	39.1MB
	SAROS	24.4KB	24.4KB	24.4KB

These approaches require prior knowledge about routing and measurement points, and cannot work in the situation out of the assumptions like node failures or routing changes.

Routing-oblivious measurement. Routing-oblivious network-wide heavy hitters detection methods have developed over the past few years. IP_Mark [2] marks unused bits in the IP packet header as 1 to indicate whether it has been sampled in other NMPs. However, marker-based methods may cause security issues, as attackers can easily avoid detection using the same rules. INVEST [12] utilizes the cardinality estimation property of Hyperloglog [14] to eliminate duplicate packet counting but does not retain flow ID-related information for heavy hitter detection. AROMA [5] addresses the issue of duplicate packets relying on a k-partition hash-based structure, which realizes uniform sampling and network-wide heavy hitter in any network topology. However, AROMA requires significant memory usage in the data plane.

6 CONCLUSION

We investigate network-wide routing-oblivious heavy hitter detection and propose SAROS (Self-Adaptive Routing Oblivious Sampling) that can ensure both accuracy and low memory occupation during the measurements. SAROS implements the KMV algorithm based on the idea of hash distance estimation, overcoming the challenges of packet double-counting problems and inaccurate measurements of the same packet in this scenario. Furthermore, self-adaptive sampling only needs to keep a dynamic threshold in the data plane, addressing the limit of storage space utilization in the programmable switch data plane. Besides, SAROS has good scalability for other measurement tasks.

ACKNOWLEDGMENTS

We thank all the reviewers for their insightful comments. This work is supported in part by National Key R&D Program of China (Grant No. 2022YFB3103000), in part by the National Natural Science Foundation of China (Grant No. 62372027 and U23B2025).

REFERENCES

[1] 2024. SAROS. https://anonymous.4open.science/r/SAROS_HHD-078E/README.md.
[2] Yehuda Afek, Anat Bremler-Barr, Shir Landau Feibish, and Liron Schiff. 2018. Detecting Heavy Flows in the SDN Match and Action Model. *Computer Networks* 136 (2018), 1–12.
[3] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. 2022. Aggregate-based congestion control for pulse-wave DDoS defense. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 693–706.

[4] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *Randomization and Approximation Techniques in Computer Science: 6th International Workshop*. 1–10.

[5] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, Shir Landau Feibish, Danny Raz, and Minlan Yu. 2020. Routing oblivious measurement analytics. In *2020 IFIP Networking Conference*. 449–457.

[6] T. Benson, A. Akella, and D. A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *ACM SIGCOMM Conference on Internet Measurement*.

[7] Kevin Beyer, Peter J Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. 2007. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. 199–210.

[8] Pierre Borgnat, Guillaume Dewaele, Kensuke Fukuda, Patrice Abry, and Kenjiro Cho. 2009. Seven years and one day: Sketching the evolution of internet traffic. In *IEEE INFOCOM 2009*. 711–719.

[9] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44 (2014), 87–95.

[10] Graham Cormode. 2009. Count-Min Sketch.

[11] Damu Ding, Marco Savi, Gianni Antichi, and Domenico Siracusa. 2020. An Incrementally-Deployable P4-Enabled Architecture for Network-Wide Heavy-Hitter Detection. *IEEE Transactions on Network and Service Management* 17 (2020), 75–88.

[12] Damu Ding, Marco Savi, Federico Pederzolli, and Domenico Siracusa. 2021. INVEST: Flow-based Traffic Volume Estimation in Data-plane Programmable Networks. In *2021 IFIP Networking Conference*. 1–9.

[13] Nick G Duffield and Matthias Grossglauser. 2001. Trajectory sampling for direct traffic observation. *IEEE/ACM transactions on networking* 9, 3 (2001), 280–292.

[14] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*. 137–156.

[15] Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. 2017. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications* 88 (2017), 50–71.

[16] Alex Graves and Alex Graves. 2012. Long Short-Term Memory. *Supervised Sequence Labelling With Recurrent Neural Networks* (2012), 37–45.

[17] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 357–371.

[18] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-Wide Heavy Hitter Detection with Commodity Switches. In *Proceedings of the Symposium on SDN Research*. 1–7.

[19] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A better netflow for data centers. In {*USENIX*} *Symposium on Networked Systems Design and Implementation*. 311–324.

[20] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 101–114.

[21] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)*. 3829–3846.

[22] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. 1994. Network intrusion detection. *IEEE network* 8, 3 (1994), 26–41.

[23] Yuan Tao and Shui Yu. 2013. DDoS attack detection at local area networks using information theoretical metrics. In *2013 12th IEEE international conference on trust, security and privacy in computing and communications*. IEEE, 233–240.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[25] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[26] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*.

[27] Huancheng Zhou, Sungmin Hong, Yangyang Liu, Xiapu Luo, Weichao Li, and Guofei Gu. 2023. Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3178–3192.

[28] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.