Hindawi
Security and Communication Networks
Volume 2022, Article ID 3271433, 18 pages
https://doi.org/10.1155/2022/3271433

Research Article

An Efficient Method for Detecting Supernodes Using Reversible Summary Data Structures in the Distributed Monitoring Systems

Aiping Zhou [1,2] **and Jin Qian** [1]

[1]*School of Information Engineering, Taizhou University, Taizhou, China*
[2]*Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China*

Correspondence should be addressed to Aiping Zhou; apzhou@njnet.edu.cn

Received 23 December 2021; Accepted 9 June 2022; Published 30 June 2022

Academic Editor: Weiwei Liu

Supernode detection has many applications in detecting network attacks, assisting resource allocation, etc. As 5G/IoT networks constantly grow, big network traffic brings a great challenge to collect massive traffic data in compact and real-time way. Previous works focus on detecting supernodes in a measurement point, while only a few works consider it in the distributed monitoring system. Moreover, they are not able to measure two types of node cardinalities simultaneously and reconstruct labels of supernodes efficiently due to large calculation and memory cost. To address these problems, we propose a novel reversible and distributed traffic summarization called RDS to simultaneously measure source and destination cardinalities for detecting supernodes in the distributed monitoring system. The basic idea of our approach is that each monitor generates a summary data structure using the coming packets and sends the summary data structure to the controller; then, the controller aggregates the received summary data structures, estimates node cardinalities, and reconstructs labels of supernodes according to the aggregated summary data structure. The experimental results based on real network traffic demonstrate that the proposed approach can detect up to 96% supernodes with a low memory requirement in comparison with state-of-the-art approaches.

1. Introduction

Traffic measurement provides valuable information for network security and network management, such as traffic accounting, load balancing, and anomaly detection [1–5], where measuring cardinality is an important task of traffic measurement. Cardinality measurement is a prerequisite for detecting supernodes, which have been paid extensive attention to by both academic and industrial organizations, despite many efforts in detecting supernodes over recent decades.

The cardinality of a node indicates the number of distinct other nodes it communicates with. We consider two types of node cardinalities: source cardinality and destination cardinality, where source cardinality indicates the number of distinct destinations that a node connects to and destination cardinality indicates the number of distinct sources that a node is connected to [6]. Supernodes are defined as the nodes whose cardinalities are more than the predefined threshold, where nodes with large source cardinality are supersources and nodes with large destination cardinality are superdestinations. A packet stream in traffic measurement can be modeled as the set of two-tuple (s, d), where s is a source which consists of some source fields from packet header, such as source address, source port, and source address and source port pair, and d is a destination which consists of some destination fields from packet header, such as destination address, destination port, and destination address and destination port pair. The problem we solve in this paper is called supernode detection, which is to report nodes whose cardinalities exceed the predefined threshold in a measurement period. For each source s, the cardinality SC (s) of s is the number of distinct destination d. If SC (s) is more than the predefined threshold of source cardinalities, s

is a supersource. Similarly, for each destination d, the cardinality DC (d) of d is the number of distinct source s, and d is a superdestination as DC (d) is more than the predefined threshold of destination cardinalities.

There are three basic tasks in traffic measurement, that is, flow size, flow persistence, and flow cardinality. Flow size is the number of elements contained in packets with the same flow label, where elements may be the entire packets, bytes in payload, and the specific content in packets. Flow persistence is the number of timeslots in which its packets occur. Flow cardinality is the number of distinct flows with the same source or destination. The measurement of flow cardinality is different from that of flow size and flow persistence. Measuring flow cardinality only counts once when the same flow appears several times, whereas measuring flow size and flow persistence needs to count the number of occurrences of one same flow. Let the measurement period be 10 minutes. Considering a case where a number of 1000 distinct hosts send 100000 packets to a server in one measurement period and these packets constitute a flow whose flow label is server address, the flow size is obviously 100000. If these packets occur in 20 timeslots, its persistence is only 20 as the measurement period is divided into 100 timeslots. Meanwhile, its cardinality is 1000, which may indicate the beginning of an attack.

Of particular importance are heavy hitters with large flow size, persistent items with large flow persistence, and supernodes with large flow cardinality. Heavy hitter detection has been extensively studied [7–14], and it has many applications, such as traffic accounting and load balancing. Persistent item detection is widely used in anomaly detection, stealthy network attack, etc. [15–18]. In this paper, we focus on supernode detection, which is a more challenging work since it is difficult to only count distinct flows appearing in one measurement period. It is important for many applications [19–24], such as DDoS attack detection and network scanning detection. For DDoS attack, the attacker makes use of infected hosts to launch a large number of requests to the targeted server in a short period, leading to consuming its resources on a large scale and making it unable to provide normal services, where the targeted server with large cardinality is a victim called superdestination. For network scanning, a malicious host attempts to connect to a large of distinct destination addresses or ports so as to discover the vulnerability in the network system, where the malicious host with large cardinality is an attacker called supersource.

There have been many efforts on cardinality estimation. A simple method is to maintain distinct connections between any two hosts in the network [25, 26]. The method can measure the cardinality of each host accurately, but it is not practical to process massive network traffic due to large memory overhead [27, 28]. Sampling-based methods [29, 30] and data stream-based methods [31–33] are proposed to tackle the challenges caused by big network traffic. However, the estimation accuracy of sampling-based methods depends on the sampling rate, that is, they maintain a small number of distinct connections at the small sampling rate, but the accuracy of cardinality estimation decreases; on the contrary, the accuracy of cardinality estimation increases at the high sampling rate, but the memory overhead also increases. Data stream-based methods demonstrate great superiority on memory utilization and measurement accuracy, where various types of summary data structures have been widely used in traffic measurement due to the excellent compression efficiency and acceptable accuracy.

However, the existing data stream-based methods still encounter challenges regarding supernode detection. First, when massive network traffic constantly generates at a high rate, they are difficult to store a large number of flows due to limited system resources on measurement points. Therefore, there need to be compact data structure by which network traffic is efficiently compressed. Second, they cannot well support supernode detection in the distributed monitoring systems, including distributed traffic collection and aggregation, centralized cardinality estimation, and supernode detection where distributed traffic collection and aggregation is to merge network traffic from multiple measurement points and centralized cardinality estimation and supernode detection are to identify supernodes based on estimated cardinality using the aggregated data structure. Since the connections established by supernodes may span the entire network, there may be a small number of its connections observed at one measurement point, while its aggregation from multiple measurement points will have a large cardinality. Hence, there is necessary a traffic collection approach, which can handle data in a distributed manner to measure node cardinalities and detect supernodes. Third, it is difficult to measure two types of node cardinalities simultaneously. Usually, they can only measure one type of node cardinality, since their summary data structures are two-dimensional bit arrays with the fixed row size or column size. Though we can measure various types of cardinalities using existing summary data structures, multiple instances of summary data structure need to be established by distinct keys, which is likely to cause excessive memory consumption. Besides, big network traffic brings great challenges to simultaneously measure two kinds of supernode cardinalities due to its one-pass requirement. Thus, it is essential to design a cardinality estimation approach for detecting supersources and destinations simultaneously. Fourth, they are unable to reconstruct labels of supernodes efficiently due to large calculation cost and unavoidable false positives and false negatives. Consequently, it is vital to reconstruct their labels for detecting supernodes efficiently and accurately.

Likewise, we also confront some challenges in detecting supernodes. First of all, it is not easy to count cardinalities of each node from multiple measurement points, since a number of connections may cross the entire network and are not simply added. In addition, to simultaneously measure two types of cardinalities for supersources and destinations, source and destination addresses need to be compressed into summary data structure once. Finally, it is very difficult to efficiently recover labels of supernodes by only summary data structure without storing source and destination addresses. These challenges in detecting supernodes motivate our work.

To solve the challenges, we propose a novel solution of supernode detection, and it obtains many flows whose cardinalities are larger than the predefined threshold at the end of one measurement period. The proposed method maps each flow to one bit of two-dimensional bit arrays, aggregates the generated two-dimensional bit arrays, estimates node cardinalities using probabilistic counting algorithm, reconstructs flow labels of supernode by inverse calculation, and detects supernodes. It mainly includes four steps: (i) update operation is used to extract flow labels (e.g., source and destination pairs) from packet stream and compress them into two-dimensional bit arrays using a group of hash functions. Then, the generated two-dimensional bit arrays are aggregated into one two-dimensional bit array with the same size at the end of each measurement period. It supports distributed traffic collection and centralized analysis. (ii) Estimation operation can be used to simultaneously measure source and destination cardinalities by only utilizing summary data structure once, and the minimal estimated cardinalities are taken as their estimations in order to mitigate the overestimation problem. (iii) Reconstruction operation is used to efficiently create supersources and destinations by inverse calculation without storing the information associated with sources and destinations. (iv) Detection operation is used to identify supersources and destinations under the guidance of abnormal rows and columns without searching the entire possible abnormal rows and columns through reconstruction operation. Our main contributions are summarized as follows.

In this paper, we design a novel reversible and distributed summary data structure to be suitable for supernode detection with accuracy and memory size guarantees in the distributed monitoring system. It is able to effectively handle a large amount of network traffic arriving by a group of hash functions, simultaneously estimate two types of source and destination cardinalities by probabilistic counting method, and efficiently detect supersources and destinations by reconstructing flow labels of supernodes based on the aggregated summary data structure. We theoretically analyze the computation, space complexity, and estimation accuracy of our method. We conduct extensive experiments on real traffic traces from WIDE to evaluate the performance of our method. The experimental results demonstrate that our method achieves superior performance compared to state-of-the-art methods in accurately and efficiently detecting supersources and destinations.

The rest of this paper is organized as follows: Section 2 summarizes related works; Section 3 formulates the problem; Section 4 presents our method and its theoretical analysis; Section 5 evaluates performance and conducts experiments; Section 6 concludes this work.

2. Related Work

Network traffic measurement has been extensively applied in many fields, where the per-flow [34–39], heavy hitter [9–12], persistent item [16, 17], and supernode measurement [19–21] are still hot topics.

Much prior work focuses on per-flow size and cardinality measurement. The task of per-flow size measurement is to count the number of elements in each flow, where flows can be TCP flows, UDP flows, or any other types defined according to the specific application requirements, and elements may be packets, bytes, or occurrences for certain events. The simple method allocates a counter for each flow to measure its size. When each packet arrives, the corresponding counter is increased by an integer, for instance, one at the packet level, the number of bytes at the byte level, or the number of accesses to websites. However, there are a large number of flows in high-speed networks, leading to enormous memory consumption. Therefore, many strategies to improve memory utilization were proposed, for instance, CAESAR [40] and virtual sketch [41], which make flows share counters or sketches to reduce memory consumption. Compared to per-flow size measurement, per-flow cardinality is difficult to be measured, since it counts the number of distinct elements in each flow. Per-flow size measurement is not able to be used in that of its cardinality directly. Most existing mechanisms were that distinct elements in each flow share the same bit vector, such as [42–44], which create a virtual bit vector for each flow, each bit of which is selected from the same bit vector using hash functions, so as to reduce memory cost.

Heavy hitter measurement is to find flows whose sizes are more than the predefined threshold in the measurement period. The universal approach is to keep track of a small set of flows, trying to retain large flows in the set while replacing small ones with new flows, such as Lossy Counting [45] and Space Saving [46]. Persistent item measurement is to find flows that occur in many timeslots. Supernode measurement is to find flows whose cardinalities are more than the predefined threshold in the measurement period, and it has been used to find attackers or victims. Supernode detection can be treated as a special case of heavy hitter detection through identifying each source or destination with a large number of connections. However, the existing solutions [47] of heavy hitter detection cannot be directly used to solve the problems of supernode identification, since they are not able to filter repetitive connections in the data streams using counters allocated to each flow. We discuss the existing solutions to detect supernodes below.

The traditional approaches maintain all distinct connections for each source or destination to detect DDoS attackers or targets in the measurement period. Although they can detect supernodes accurately, they cause great memory usage due to a large number of flows in high-speed networks.

Flow sampling-based approaches are used to monitor a set of flows whose hash values are smaller than the predefined sampling rate [48]. Therefore, the flows with many connections are very probable to be sampled. Flow sampling-based approaches improve memory efficiency, but the accuracy of supernode identification depends on the sampling rate. Moreover, they maintain the sampled flows at high calculation and memory access cost.

Data streaming-based approaches are used to detect supernodes. Most existing solutions usually design summary

data structures that fit in fast memory, and they encode flow labels extracted from arriving packets to be stored in the summary data structures. However, they cannot recover supernodes merely using the summary data structures in fast memory due to their irreversibility. Sketches are one type of summary data structures, which are designed to detect supernodes and solve irreversibility [49]. Wang et al. [20] proposed a double connection degree sketch (DCDS) that is used to reconstruct host addresses with large cardinalities based on Chinese Remainder Theorem. Liu et al. [21] designed a vector bloom filter for supernode detection, which extracts several bits directly from flow labels. However, the calculation cost is obvious for large address space.

Some network-wide measurement systems solve the problem of supernode detection [50, 51]. The proposed summary data structures can be a component of network-wide measurement systems.

Besides, some variants of supernode detection are proposed in the literature [52, 53]. Zhou et al. [52] proposed the solution of persistent spread problem, which counts the number of distinct elements in each flow persistently occurring in the predefined measurement periods. Huang et al. [53] further solved the k-persistent spread problem, which measures the number of distinct elements in each flow appearing in at least k out of t measurement periods.

3. Problem Formulation

In this paper, we consider a distributed monitoring system composed of the controller and a set of monitors, as shown in Figure 1. Let $P = p_1, p_2, \ldots, p_t, \ldots$ be a sequentially arriving packet stream generated by network traffic packets, where $p_t = (s_t, d_t)$ is some fields of the tth packet, in which s_t and d_t are the corresponding source and destination, respectively. Source or destination space consists of distinct source or destination over one measurement period, which are denoted as S and D. A source can be any combination of source fields in the packet header, such as source IP, source port, or their combination. Similarly, a destination can be any combination of destination fields in the packet header, such as destination IP, destination port, or their combination. The source and destination are determined according to the specific application requirement. In this work, we use the source and destination pair of one packet as its flow label. The whole measurement time is partitioned into many measurement periods T with equal length. At the beginning of one measurement period, the monitor extracts some fields from the arriving packets and compresses them into summary data structures. At the end of one measurement period, the monitor sends the generated data structures to the controller and resets data structures. Then, the controller aggregates the received data structures and detects supernodes.

For any source $s \in S$, its cardinality is defined as the number of distinct destinations that s connects to in one measurement period. Similarly, for any destination $d \in D$, its cardinality is defined as the number of distinct sources that connects to d in one measurement period. Supernodes are divided into two types: supersources and superdestinations.

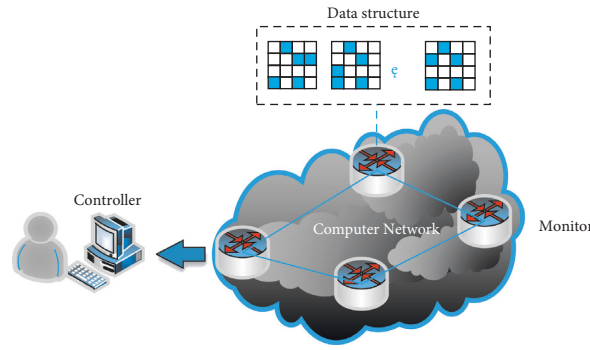

Figure 1: The distributed monitoring system.

Supersources or superdestinations are the hosts whose cardinality exceeds the predefined threshold $\theta_1 D_1$ or $\theta_2 D_2$ in one measurement period, where D_1 or D_2 denotes the sum of source or destination cardinality in one measurement and θ_1 and θ_2 are constants, $0 < \theta_1$ and $\theta_2 < 1$. Supersources and superdestinations are expressed as

$$\mathrm{SS} = \{s | \mathrm{SC}(s) > \theta_1 D_1, \quad s \in S\}, \tag{1}$$

$$\mathrm{SD} = \{d | \mathrm{DC}(d) > \theta_2 D_2, \quad d \in D\}, \tag{2}$$

where SC (s) indicates the cardinality of source s, DC (d) indicates the cardinality of destination d, D_1 is computed as $\sum_{s \in S} \mathrm{SC}(s)$, and D_2 is computed as $\sum_{d \in D} \mathrm{DC}(d)$.

Supernode detection is widely used in many areas. For example, port scanning attacks are performed by trying to connect to numerous distinct destination addresses or ports for the existence of vulnerable services. In this case, the attacker with large cardinality is a supersource. Besides, Distributed Denial-of-Service (DDoS) attacks are launched by using a large number of connected devices as attackers to send a lot of requests to a victim such as server, so that legitimate users are not able to utilize its resources. Similarly, the victim with high cardinality is a superdestination.

The goal of this work is to design a distributed monitoring framework which consists of updating module and detecting module. The former stores information associated with cardinalities of flows by summary data structures on each monitor. The latter aggregates the generated data structures from each monitor, estimates source and destination cardinality, reconstructs supersource and destination candidates, and identifies supersources and destinations on the controller. We perform theoretical analysis and performance evaluation.

4. Our Algorithm

In this section, we first introduce a novel summary data structure. Then, we define the main operations of our algorithm, containing updating and aggregating summary data structures, estimating cardinality, reconstructing sources and destinations, and detecting supersources and superdestinations. Theoretical analysis on updating complexity and estimation accuracy is performed. Next, we elaborate them in detail. The framework of our method is shown in Figure 2.

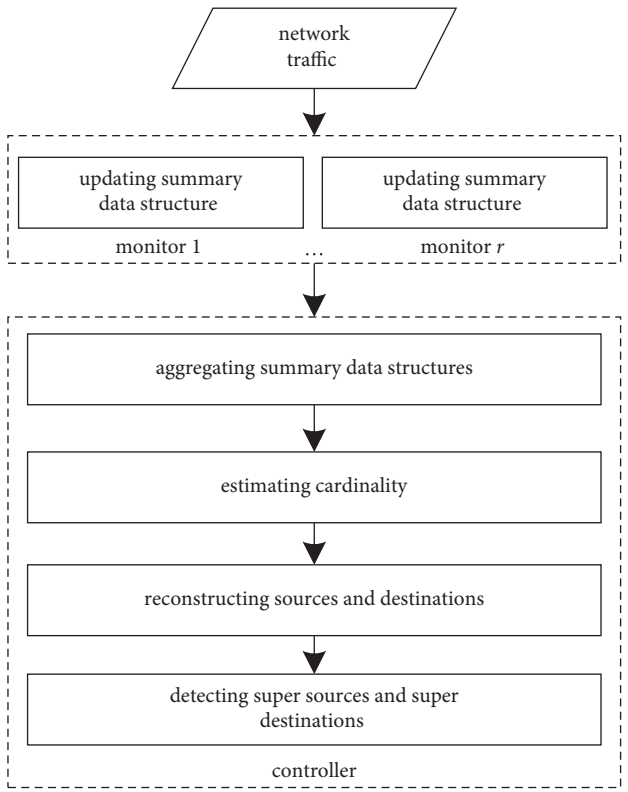

Figure 2: Framework of our method.

4.1. Data Structure. The summary data structure is denoted as follows:

$$B = (B_1, B_2, \ldots, B_H). \tag{3}$$

Each B_i $(1 \le i \le H)$ is a two-dimensional bit array with the size of $n_i \times m_i$, each bit of which is denoted as $B_i[j][k]$ $(0 \le j \le n_i - 1$ and $0 \le k \le m_i - 1)$, as shown in Figure 3. Each B_i $(1 \le i \le H)$ with different sizes is used to store sufficient flow information and effectively trace attackers or victims. We use a row hash function f_i and a column hash function h_i to locate the index in each B_i $(1 \le i \le H)$. The row and column hash functions are expressed as

$$f_i\colon \{0, 1, \ldots, N-1\} \longrightarrow \{0, 1, \ldots, n_i - 1\}, \tag{4}$$

$$h_i\colon \{0, 1, \ldots, M-1\} \longrightarrow \{0, 1, \ldots, n_i - 1\}, \tag{5}$$

where N and M are the size of source space and destination space and n_i and m_i indicate the number of rows and columns in each B_i $(1 \le i \le H)$.

To recover abnormal sources by simple computation, the row hash function f_i and column hash function h_i are defined as

$$f_i(x) \equiv c_i \pmod{n_i}, \quad 1 \le i \le H, \tag{6}$$

$$h_i(x) \equiv c_i' \pmod{m_i}, \quad 1 \le i \le H, \tag{7}$$

where c_i and c_i' are the values of modulus operation, n_1, n_2, …, n_H are selected as pair-wise coprime integers to make the summary data structure reversible, and m_1, m_2, …, m_H are also pair-wise coprime integers.

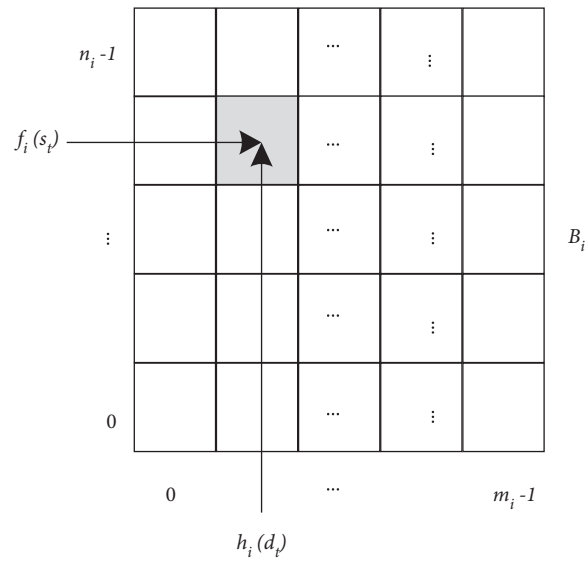

Figure 3: Data structure.

The monitors use the same summary data structure B as the controller. Let the number of monitors be R. The summary data structure on the monitors is indicated as follows:

$$B^r = (B_1^r, B_2^r, \ldots, B_H^r), \quad r = 1, 2, \ldots, R. \tag{8}$$

4.2. Updating Summary Data Structures. Updating operation is used to collect flow information from massive network traffic. At the beginning of one measurement period, all bits in B and each B^r $(1 \le r \le R)$ are initialed. Then, each monitor updates the corresponding bit in each B_i^r $(1 \le i \le H)$ by the row hash function f_i $(1 \le i \le H)$ and column hash function h_i $(1 \le i \le H)$ when a packet arrives. At the end of one measurement period, the controller aggregates the generated summary data structures from each monitor. The updating process is described in Algorithm 1.

When packet stream arrives sequentially, each monitor extracts its flow label (s_t, d_t) from one packet and sets the corresponding bit in each B_i^r $(1 \le i \le H, 1 \le r \le R)$ by the row hash function f_i (s_t) $(1 \le i \le H)$ and column hash function h_i (d_t) $(1 \le i \le H)$ to one, which is denoted as follows:

$$B_i^r[f_i(s_t)][h_i(d_t)] = 1, \quad 1 \le r \le R, 1 \le i \le H. \tag{9}$$

At the end of one measurement period, each monitor sends the generated summary data structure to the controller. Then, the controller performs the bitwise-OR operations on the same bits in each B_i^r $(1 \le i \le H$ and $1 \le r \le R)$. If one bit in $B_i^r[j][k]$ $(1 \le r \le R)$ is one, the corresponding bit in each B_i $(1 \le i \le H)$ in B is set to one. The bitwise-OR operation is denoted as follows:

$$B_i[j][k] = B_i^1[j][k] \oplus \cdots \oplus B_i^R[j][k], \tag{10}$$

Input: initialize $B^r = (B_1^r, B_2^r, \ldots, B_H^r)$, $1 \le r \le R$
Output: the updated $B = (B_1, B_2, \ldots, B_H)$
(1) **for** each packet arriving at the monitor r **do**
(2) extract its flow label (s_t, d_t)
(3) compute $f_i(s_t)$, $1 \le i \le H$
(4) compute $h_i(d_t)$, $1 \le i \le H$
(5) $B_i^r[f_i(s_t)][h_i(d_t)] \leftarrow 1$, $1 \le i \le H$, $1 \le r \le R$
(6) **end for**
(7) $B_i[j][k] \leftarrow B_i^1[j][k] \oplus \cdots \oplus B_i^R[j][k]$, $1 \le i \le H$

Algorithm 1: Updating summary data structures.

Input: the updated $B = (B_1, B_2, \ldots, B_H)$
Output: the estimated source and destination cardinality
(1) **for** one source $s \in S$ **do**
(2) compute $f_i(s)$, $1 \le i \le H$
(3) $B_i(s) \longleftarrow B_i[f_i(s)][\cdot]$, $1 \le i \le H$
(4) compute the number $U_{B_i(s)}$ of zero bits in each $B_i(s)$
(5) $\mathrm{SC}_i(s) \longleftarrow -n_i \ln(U_{B_i(s)}/n_i)$, $1 \le i \le H$
(6) $\mathrm{SC}(s) \longleftarrow \min_{i=1}^{N} \mathrm{SC}_i(s)$
(7) **end for**
(8) **for** one destination $d \in D$ **do**
(9) compute $h_i(d)$, $1 \le i \le H$
(10) $B_i(d) \longleftarrow B_i[\cdot][h_i(d)]$, $1i \le i \le H$
(11) compute the number $U_{B_i(d)}$ of zero bits in each $B_i(d)$
(12) $\mathrm{DC}_i(d) \longleftarrow -m_i \ln(U_{B_i(d)}/m_i)$, $1 \le i \le H$
(13) $\mathrm{DC}(d) \longleftarrow \min_{i=1}^{N} \mathrm{DC}_i(d)$
(14) **end for**

Algorithm 2: Estimating cardinality.

where $B_i^r[j][k]$ $(1 \le r \le R)$ indicates the corresponding bit in the ith bit array B_i^r in B^r and $\oplus$ is a bitwise-OR operator.

4.3. Estimating Source and Destination Cardinality. Estimating operation is used to obtain an approximate estimation of cardinality based on the aggregated data structure $B = (B_1, B_2, \ldots, B_H)$. Estimating operation is shown in Algorithm 2. For each source $s \in S$, we compute the hash value $f_i(s)$ of source s to locate the row in each B_i $(1 \le i \le H)$ of B. The flows associated with source s are mapped to H rows $B_i(s) = B_i[f_i(s)][\cdot]$ $(1 \le i \le H)$. As a result, we obtain H bit vectors $B_i(s)$ $(1 \le i \le H)$ to store the cardinality information of source s. The source cardinality for each bit vector $B_i(s)$ $(1 \le i \le H)$ is estimated as (11) by the probabilistic counting algorithm.

$$SC_i(s) = -n_i \ln \frac{U_{B_i(s)}}{n_i}, \tag{11}$$

where $U_{B_i(s)}$ is the number of zero bits in each bit vector $B_i(s)$ $(1 \le i \le H)$.

If other sources are not mapped to the $f_i(s)$th row, the estimated source cardinality is very close to its real value. Actually, each $B_i(s)$ $(1 \le i \le H)$ may contain noise caused by other sources, so that the source cardinality may be overestimated. Therefore, we use the minimum value of estimated source cardinalities $\mathrm{SC}_i(s)$ $(1 \le i \le H)$ as its estimation [40], which is denoted as follows:

$$\mathrm{SC}(s) = \min_{i=1}^{N} \mathrm{SC}_i(s). \tag{12}$$

Similarly, for each destination $d \in D$, we calculate the hash value $h_i(d)$ of destination d to locate the column in each B_i $(1 \le i \le H)$ of B. The flows associated with destination d are hashed to H bit vectors $B_i(d) = B_i[\cdot][h_i(d)]$ $(1 \le i \le H)$. Therefore, the destination cardinality for each bit vector $B_i(d)$ $(1 \le i \le H)$ is estimated as (13) by the probabilistic counting algorithm.

$$\mathrm{DC}_i(d) = -m_i \ln \frac{U_{B_i(d)}}{m_i}, \tag{13}$$

where $U_{B_i(d)}$ is the number of zero bits in each bit vector $B_i(d)$ $(1 \le i \le H)$.

We use the minimum value of estimated destination cardinalities $\mathrm{DC}_i(d)$ $(1 \le i \le H)$ as its estimation, which is denoted as follows:

$$\mathrm{DC}(d) = \min_{i=1}^{N} \mathrm{DC}_i(d). \tag{14}$$

4.4. Reconstructing Sources and Destinations. Reconstructing operation is to recover abnormal sources and destinations. For ease of understanding, we first

Input: the updated $B = (B_1, B_2, \ldots, B_H)$
Output: supersources and destinations
(1) **for** $i = 1$ to H **do**
(2) compute the number $U_{B_i[j][\cdot]}$ of zero bits in each B_i $[j]$ $[\cdot]$
(3) $\mathrm{RC}_i^j \leftarrow -n_i \ln(U_{B_i[j][\cdot]}/n_i)$, $0 \le j \le n_i - 1$
(4) **if** $\mathrm{RC}_i^j > \alpha_i$ **then**
(5) the row B_i $[j]$ $[\cdot]$ is an abnormal row
(6) **end if**
(7) compute the number $U_{B_i[\cdot][j]}$ of zero bits in each B_i $[\cdot]$ $[j]$
(8) $\mathrm{CC}_i^j \leftarrow -m_i \ln(U_{B_i[\cdot][j]}/m_i)$, $0 \le j \le m_i - 1$
(9) **if** $\mathrm{CC}_i^j > \beta_i$ **then**
(10) the column B_i $[\cdot]$ $[j]$ is an abnormal column
(11) **end if**
(12) **end for**
(13) Obtain supersources and destinations by inverse calculation

Algorithm 3: Detecting supersources and superdestinations.

consider the simple situation. Suppose that there is only one abnormal row in each B_i $(1 \le i \le H)$, which is denoted as c_i $(1 \le i \le H)$. According to the predefined row hash function f_i, we can map a source s to the f_i (s)th row in each B_i $(1 \le i \le H)$, that is, $f_i(s) \equiv c_i \pmod{m_i}$ $(1 \le i \le H)$. The problem of finding abnormal source s is converted to the solution of equations $f_i(s) \equiv c_i \pmod{m_i}$ $(1 \le i \le H)$. Based on the Chinese Remainder Theorem (CRT) [44], the solutions are denoted as follows:

$$s \equiv \sum_{i=1}^{H} M_i M_i^{-1} c_i \pmod{M}, \tag{15}$$

where $M = m_1 m_2 \cdots m_H$, $M_i = M/m_i$, and $M_i M_i^{-1} \equiv 1 \pmod{m_i}$.

Similarly, we assume only one abnormal column in each B_i $(1 \le i \le H)$, which is denoted as c_i' $(1 \le i \le H)$. The problem of finding abnormal destination d is converted to the solution of equations $h_i(d) \equiv c_i' \pmod{n_i}$ $(1 \le i \le H)$. Therefore, the solutions are expressed as follows using the CRT:

$$d \equiv \sum_{i=1}^{H} N_i N_i^{-1} c_i' \pmod{N}, \tag{16}$$

where $N = n_1 n_2 \cdots n_H$, $N_i = N/n_i$, and $N_i N_i^{-1} \equiv 1 \pmod{n_i}$.

For the general situation, we assume w abnormal rows in each B_i $(1 \le i \le H)$. There are wH combinations consisting of one abnormal row or column in each B_i $(1 \le i \le H)$. We use the CRT to solve the reversible problem for each combination. The entire abnormal sources or destinations are the union of solutions with each combination. However, the reverse calculations cause large computational overhead and increase false positive rate and false negative rate due to a few false combinations and hash collisions.

We design a strategy to establish relations between two consecutive rows to which sources are mapped by row hash functions. Taking one source s as example, we obtain the row index f_i (s) $(1 \le i \le H)$ and the next row index f_{i+1} (s) $(1 \le i \le H-1)$ using row hash function f_i $(1 \le i \le H)$. Therefore, the row index $f_i(s)$ is associated with the next row index f_{i+1} (s) $(1 \le i \le H-1)$ by one hash table, that is, $T_i[f_i(s)] = f_{i+1}$ (s) $(1 \le i \le H-1)$. However, different sources may be mapped to the same rows, leading to hash collisions. Assuming another source s', $T_i[f_i(s)] = \{f_{i+1}(s), f_{i+1}(s')\}$ $(1 \le i \le H-1)$. We conduct row combinations based on the hash table to reduce computational overhead and improve false positive rate and false negative rate. After that, we can accurately recover sources from row combinations using inverse calculation. Similarly, supposing that there are two destinations d and d' mapped to the same columns, $T_i[h_i(d)] = \{h_{i+1}(d), h_{i+1}(d')\}$ $(1 \le i \le H-1)$. Likewise, we conduct column combinations based on the hash table and recover destinations using inverse calculation to reduce computational overhead and improve false positive rate and false negative rate.

As a result, it reduces false positive rate generated by false row combinations. Supersources and destinations show abnormal rows and columns at high probability.

4.5. Detecting Supersources and Superdestinations. For detecting supersources, we should identify abnormal rows generated by supersources at high probability in each B_i $(1 \le i \le H)$. We view each row $B_i[j][\cdot]$ $(1 \le i \le H, 0 \le j \le n_i - 1)$ as one bit vector. For each row B_i $[j]$ $[\cdot]$ $(1 \le i \le H, 0 \le j \le n_i - 1)$, its cardinality is estimated as (17) by the probabilistic counting algorithm.

$$\mathrm{RC}_i^j = -n_i \ln \frac{U_{B_i[j][\cdot]}}{n_i}, \tag{17}$$

where $U_{B_i[j][\cdot]}$ is the number of zero bits in each row B_i $[j]$ $[\cdot]$ $(1 \le i \le H, 0 \le j \le n_i - 1)$. If the cardinality RC_i^j of each row B_i $[j]$ $[\cdot]$ $(1 \le i \le H, 0 \le j \le n_i -)$ is more than the predefined threshold α_i which is the ratio of summation of source cardinalities in a measurement period, the row B_i $[j]$ $[\cdot]$ $(1 \le i \le H, 0 \le j \le n_i - 1)$ is defined as one abnormal row. Similarly, for detecting superdestinations, we should identify abnormal columns caused by superdestinations at high probability in each B_i $(1 \le i \le H)$. For each column B_i $[\cdot]$ $[j]$ $(1 \le i \le H, 0 \le j \le n_i -)$, its cardinality is estimated as (18) by the probabilistic counting algorithm.

$$\mathrm{CC}_i^j = -m_i \ln \frac{U_{B_i[\cdot][j]}}{m_i}, \tag{18}$$

where $U_{B_i[\cdot][j]}$ is the number of zero bits in each column $B_i[\cdot][j]$ $(1 \le i \le H,\ 0 \le j \le m_i - 1)$. If the cardinality CC_i^j of each column $B_i[\cdot][j]$ $(1 \le i \le H,\ 0 \le j \le m_i - 1)$ is more than the predefined threshold β_i which is the ratio of summation of destination cardinalities, the column $B_i[\cdot][j]$ $(1 \le i \le H,\ 0 \le j \le m_i - 1)$ is defined as one abnormal column.

After that, we obtain abnormal rows and columns in each B_i $(1 \le i \le H)$. Therefore, supersources and superdestinations can be identified by reconstructing operation. The detection operation is shown in Algorithm 3.

4.6. Theoretical Analysis. In the updating process, there need to be $2H$ hash calculations to determine the row and column in H two-dimensional bit arrays and $2H$ memory accesses for each packet. For the aggregation operation, it executes $H + 2$ memory accesses. Therefore, the time complexity to update a packet is $O(H)$. For simplicity, we only consider the size of summary data structures. Each monitor needs $m_i n_i H$ memory to store the related cardination information, and the controller needs the same memory size to detect supernodes. Since the distributed monitoring system consists of the controller and multiple monitors, the required memory space is $m_i n_i$ $(H + 1)$. Therefore, the space complexity is $O(m_i n_i H)$.

We only derive the deviation and standard error of source cardinality estimation $\widehat{\mathrm{SC}}_i$ in each two-dimensional bit array B_i $(1 \le i \le H)$ according to the linear-time probabilistic counting algorithm as follows.

Let $V_{B_i(s)} = U_{B_i(s)}/n_i$; the estimation of source cardinality in each two-dimensional bit array B_i $(1 \le i \le H)$ is denoted as follows:

$$\widehat{\mathrm{SC}}_i = -n_i \ln V_{B_i(s)}. \tag{19}$$

The Taylor series of $\ln V_{B_i(s)}$ at $V_{B_i(s)} = e^{-\mathrm{SC}_i/n_i}$ is expressed as follows:

$$\ln V_{B_i(s)} = -\frac{\mathrm{SC}_i}{n_i} + \frac{V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}}{e^{-\mathrm{SC}_i/n_i}} - \frac{1}{2} \cdot \frac{\left(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}\right)^2}{e^{-2\,\mathrm{SC}_i/n_i}} + \cdots. \tag{20}$$

We take the first three items of (20), and the estimation of source cardinality in each two-dimensional bit array B_i $(1 \le i \le H)$ is approximately represented as follows:

$$\widehat{\mathrm{SC}}_i = n_i \left(\frac{\mathrm{SC}_i}{n_i} - \frac{V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}}{e^{-\mathrm{SC}_i/n_i}} + \frac{1}{2} \cdot \frac{\left(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}\right)^2}{e^{-2\,\mathrm{SC}_i/n_i}} \right). \tag{21}$$

The mathematical expectation of source cardinality estimation is denoted as follows:

$$E\left(\widehat{\mathrm{SC}}_i\right) = \mathrm{SC}_i + \frac{1}{2} \cdot \frac{n_i E\left(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}\right)^2}{e^{-2\,\mathrm{SC}_i/n_i}}. \tag{22}$$

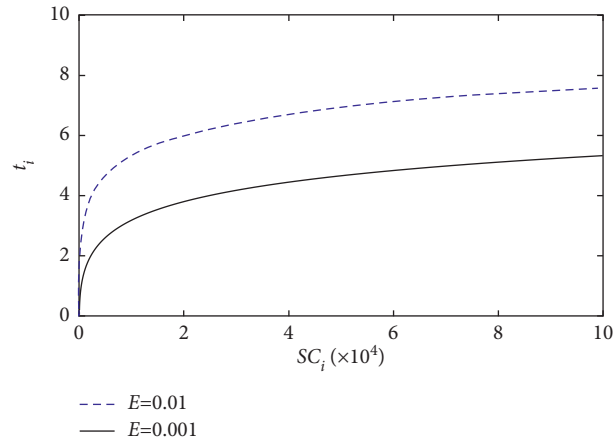

Figure 4: The parameters SC_i and t_i impact on the relative error of source cardinality estimation.

Since $E(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i})^2 = 1/n_i e^{-\mathrm{SC}_i/n_i}(1 - (1 + \mathrm{SC}_i/n_i)e^{-\mathrm{SC}_i/n_i})$, we obtain

$$E\left(\widehat{\mathrm{SC}}_i\right) = \mathrm{SC}_i + \frac{e^{\mathrm{SC}_i/n_i} - \mathrm{SC}_i/n_i - 1}{2}. \tag{23}$$

The mathematical expectation of relative error is denoted as follows:

$$E\left(\frac{\widehat{\mathrm{SC}}_i - \mathrm{SC}_i}{\mathrm{SC}_i}\right) = \frac{e^{\mathrm{SC}_i/n_i} - \mathrm{SC}_i/n_i - 1}{2\,\mathrm{SC}_i}. \tag{24}$$

Let $r_i = \mathrm{SC}_i/n_i r_i = \mathrm{SC}_i/n_i$. Equation (24) is transformed as follows:

$$E\left(\frac{\widehat{\mathrm{SC}}_i - \mathrm{SC}_i}{\mathrm{SC}_i}\right) = \frac{e^{t_i} - t_i - 1}{2\,\mathrm{SC}_i}. \tag{25}$$

Figure 4 illustrates the relationship among these parameters, namely, the source cardinality SC_i, the ratio t_i, and the relative error. When the source cardinality SC_i is constant, we can see that the relative error decreases as the ratio t_i decreases. Therefore, we can select the column number n_i of summary data structure to obtain the required relative error for each SC_i.

To derive the variance of the ratio $\widehat{\mathrm{SC}}_i/\mathrm{SC}_i$, we take the first two items of (20) as the approximate estimation of source cardinality in each two-dimensional bit array B_i $(1 \le i \le H)$, which is represented as follows:

$$\widehat{\mathrm{SC}}_i = n_i \left(\frac{\mathrm{SC}_i}{n_i} - \frac{V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}}{e^{-\mathrm{SC}_i/n_i}} \right). \tag{26}$$

The variance of the ratio $\widehat{\mathrm{SC}}_i/\mathrm{SC}_i$ is denoted as follows:

$$\mathrm{Var}\left(\frac{\widehat{\mathrm{SC}}_i}{\mathrm{SC}_i}\right) = \frac{n_i^2 \mathrm{Var}\left(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}\right)}{\mathrm{SC}_i^2 e^{-2\,\mathrm{SC}_i/n_i}}. \tag{27}$$

Since $\mathrm{Var}(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i}) = E(V_{B_i(s)} - e^{-\mathrm{SC}_i/n_i})^2$, we obtain (28) according to the previous formula:

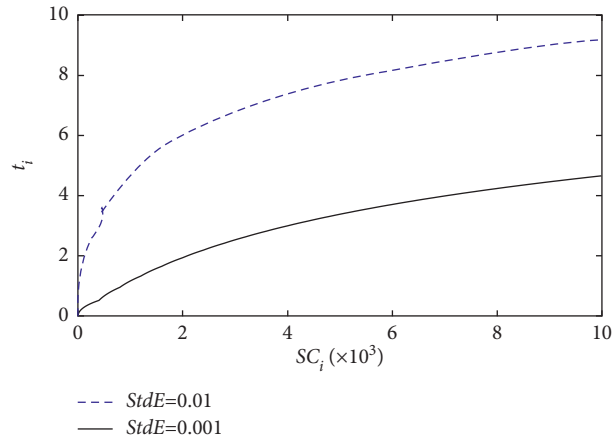

Figure 5: The parameters SC_i and t_i impact on the standard error of the ratio $\widehat{SC}_i/SC_i$.

Table 1: Statistics of datasets.

Dataset	#packet	|SIP|	|DIP|	|(SIP, DIP)|
data1	2.4M	39K	212K	658K
data2	2.4M	39K	212K	660K
data3	2.3M	39K	215K	711K

$$\mathrm{Var}\left(\frac{\widehat{\mathrm{SC}}_i}{\mathrm{SC}_i}\right) = \frac{n_i\left(e^{\mathrm{SC}_i/n_i} - \mathrm{SC}_i/n_i - 1\right)}{\mathrm{SC}_i^2}. \tag{28}$$

Therefore, the standard error of the ratio $\widehat{\mathrm{SC}}_i/\mathrm{SC}_i$ is denoted as follows:

$$\mathrm{StdE}\left(\frac{\widehat{\mathrm{SC}}_i}{\mathrm{SC}_i}\right) = \frac{\left(n_i e^{\mathrm{SC}_i/n_i} - n_i - \mathrm{SC}_i\right)^{1/2}}{\mathrm{SC}_i}. \tag{29}$$

Let $r_i = \mathrm{SC}_i/n_i$; equation (11) is transformed as follows:

$$\mathrm{StdE}\left(\frac{\widehat{\mathrm{SC}}_i}{\mathrm{SC}_i}\right) = \frac{\sqrt{n_i}\left(e^{t_i} - t_i - 1\right)^{1/2}}{\mathrm{SC}_i}. \tag{30}$$

Figure 5 shows the relationship among these parameters, namely, the source cardinality SC_i, the standard error of the ratio $\widehat{SC}_i/SC_i$, and the ratio t_i. When the source cardinality SC_i is constant, we can also see that the standard error of the ratio $\widehat{SC}_i/SC_i$ decreases as the ratio t_i decreases. Therefore, we can select the column number n_i of summary data structure to obtain the required standard error for each SC_i.

5. Experiment and Evaluation

In this section, we evaluate the performance of our algorithm in comparison with other ones. The experiments are extensively conducted on real traffic data. All algorithms are implemented using C++ on a server with Intel E-2224 CPU and 32 GB memory. The influence of parameters on the algorithm performance is discussed.

Table 2: True supersources and destinations.

Dataset	|SS|	|SD|
data1	91	98
data2	91	72
data3	83	77

5.1. Datasets. To evaluate the performance of our algorithm, we select the traffic trace in the first three minutes from traffic traces without packet header over 15 minutes published by MAWI [54], which are divided into three traffic traces with one minute called data1, data2, and data3, respectively. Table 1 shows the statistical information of three traffic traces used in our experiments, where #packet denotes the number of packets in the traffic trace, |SIP| denotes the number of distinct source IP addresses (SIP for short), |DIP| denotes the number of distinct destination IP addresses (DIP for short), and |(SIP, DIP)| denotes the number of distinct source IP address and destination IP address pairs. As shown in Table 1, the average number of packets, source IP addresses, destination IP addresses, and source IP address and destination IP address pairs in three traffic traces is 2.4M, 39K, 212K, and 676K.

In this paper, we use SIP and DIP as sources and destinations, respectively. Figure 6 shows the cardinality distribution in three traffic traces, where the x-coordinates indicate the logarithm of source or destination cardinality and the y-coordinates indicate the logarithm of number of sources or destinations. We can see that the number of sources or destinations decreases as source or destination cardinality increases. As shown in Figures 6(a), 6(c), and 6(e), the sources with low cardinality are obviously more

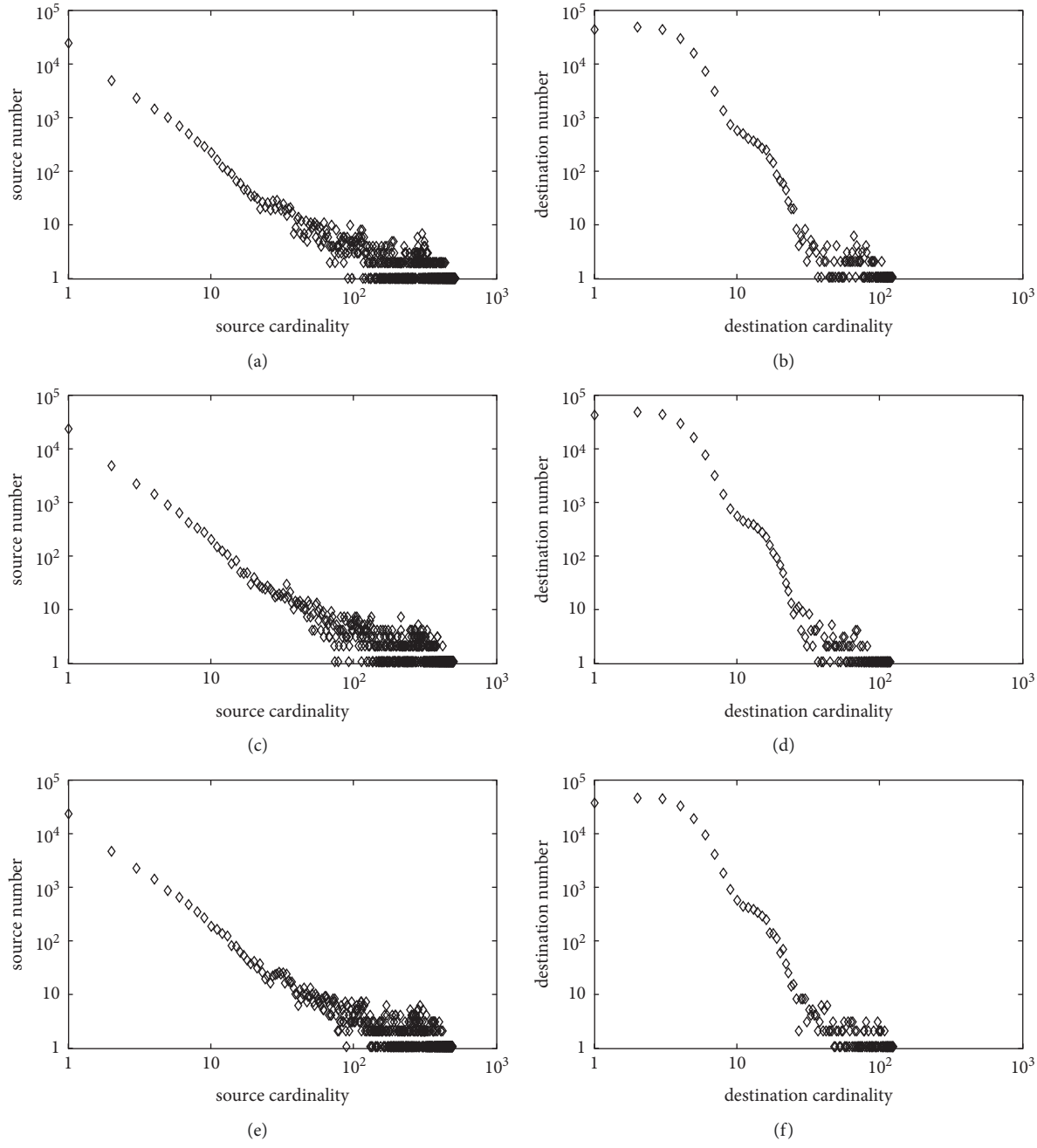

Figure 6: Cardinality distribution. (a) Distribution of source cardinality under data1. (b) Distribution of destination cardinality under data1. (c) Distribution of source cardinality under data2. (d) Distribution of destination cardinality under data2. (e) Distribution of source cardinality under data3. (f) Distribution of destination cardinality under data3.

than ones with high cardinality. The number of sources whose cardinality is less than 10 is about 93 percent of overall sources. As shown in Figures 6(b), 6(d), and 6(f), we obtain similar results, that is, the destinations with low cardinality are apparently more than ones with high cardinality. The destinations whose cardinality is less than 10 are about 98 percent of overall destinations. The cardinality approximatively obeys the heavy-tailed distribution.

In the experiments, we assume that there are three monitors in distributed monitoring systems, each of which processes the 20-second traffic trace for each traffic trace. For detecting supersources and destinations, we evaluate the performance of our algorithm compared with the compact spread estimator (CSE) [42], double connection degree sketch (DCDS) [20], and SpreadSketch (SS) [49] in terms of estimation accuracy, detection precision, and memory cost. The CSE constructs a virtual bit vector from the shared one-dimensional bit array for each host by a group of hash functions. It can provide good accuracy in a small memory. The DCDS constructs multiple two-dimensional bit arrays, only sets several bits selected in a bit array for each coming packet by a group of hash functions, and then reconstructs

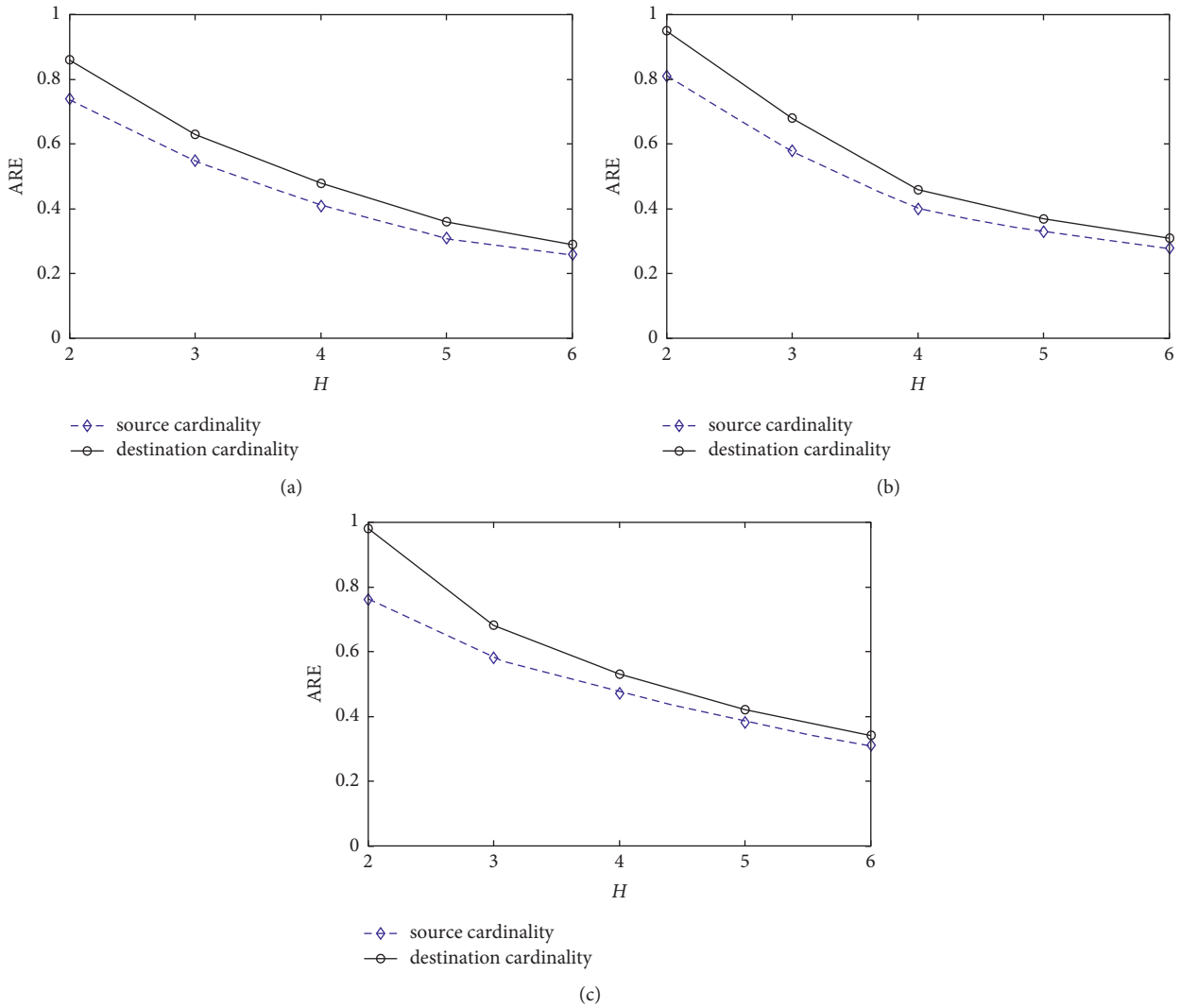

Figure 7: The influence of parameter H on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

abnormal hosts by simple inverse calculation. The SS constructs an invertible sketch that is formed by the combination of count-min sketch and multiresolution bitmap, and multiple sketches can be merged to provide a network-wide measurement view for recovering superspreaders and their estimated fan-outs by simple computations and small memory.

We also need to know true supersources and destinations obtained using three traffic traces in advance. Table 2 shows the number of true supersources and destinations when the predefined thresholds are 0.1 and 0.01 percent of the overall source and destination cardinality for three traffic traces, where |SS| and |SD| denote the number of true supersources and destinations separately.

5.2. Influence of Parameters on Estimation Accuracy. Our algorithm has three parameters, H, m_i, and n_i, where H denotes the number of hash functions and n_i and m_i denote the number of rows and columns in two-dimensional bit arrays B_i ($1 \le i \le H$). Updating, estimating, and reconstructing processes use hash functions, and the parameter H impacts the processing time. H changes from 2 to 6. Both n_i and m_i determine the size of memory consumed by our algorithm. According to the coupon collection issue, the source and destination cardinality can be accurately estimated when they are less than $n_i \ln n_i$ and $m_i \ln m_i$. In the experiments, source IP addresses and destination IP addresses are used as sources and destinations. Therefore, n_i is a prime from 400 to 800 and m_i is also a prime from 3000 to 7000.

We evaluate the accuracy of cardinality estimation by the average relative error (ARE for short), which is the mean value of difference between the true cardinality of hosts and their cardinality estimated divided by the true cardinality. ARE is expressed as follows:

$$\text{ARE} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left| \widehat{c}_i - c_i \right|}{c_i}, \tag{31}$$

where c_i denotes the true cardinality of host i, $\widehat{c}_i$ denotes the estimated cardinality of host i, and n indicates the number of

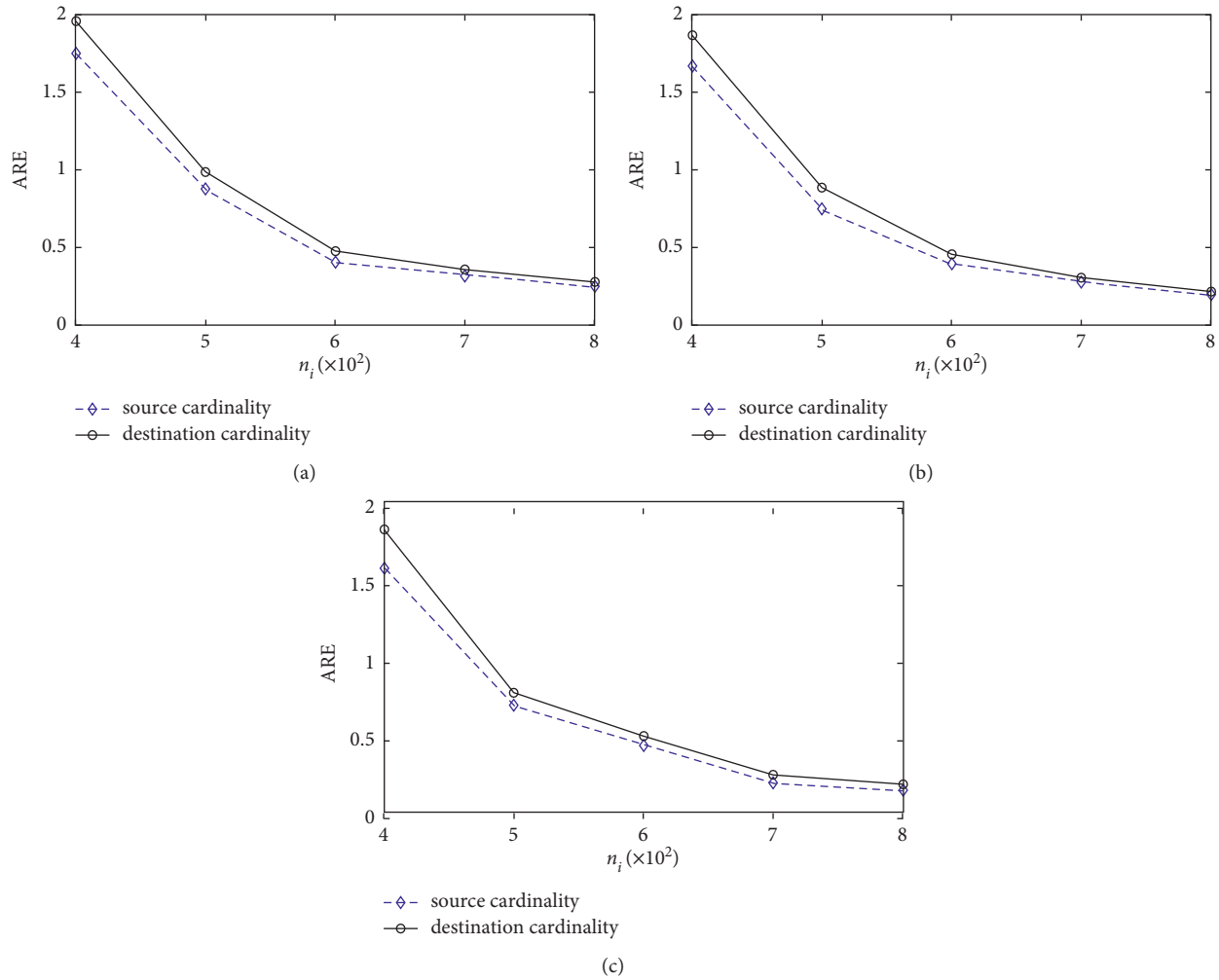

Figure 8: The influence of parameter n_i on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

distinct hosts. The smaller the ARE is, the more accurate the estimated cardinality is.

Figure 7 shows the influence on the performance of our algorithm under different traffic traces. As shown in Figure 7(a), we can see that the ARE of our algorithm decreases as H increases. As shown in Figures 7(b) and 7(c), we obtain similar results. Since distinct sources and destinations are mapped to two-dimensional bit arrays by many hash functions, hash collisions are significantly reduced. Besides, the estimation error caused by hash collisions is reduced because the minimum estimation in each two-dimensional bit array is used as the cardinality estimation.

Figure 8 shows the influence of parameter n_i on the estimation accuracy under different traffic traces. As shown in Figure 8(a), we can see that the ARE obviously decreases as n_i varies from 400 to 600 and slowly decreases as n_i changes from 600 to 800. As shown in Figures 8(b) and 8(c), we obtain similar results. When other parameters are fixed, the size of memory consumed increases as n_i increases. Therefore, the probability that distinct sources are mapped to the same bits in two-dimensional bit arrays are reduced to improve the estimation accuracy.

Based on the above analysis, we select $(H) = 4$, $n_i = 600$, and $m_i = 5000$ in the experiments.

Figure 9 shows the influence of parameter m_i on the estimation accuracy under different traffic traces. As shown in Figure 9(a), we can see that the ARE obviously decreases as m_i varies from 3000 to 5000 and slowly decreases as m_i changes from 5000 to 7000 using data1. As shown in Figures 9(b) and9(c), we obtain the similar results. Since the size of memory consumed increases, the probability of hash collisions is reduced. Therefore, we improve the cardinality estimation accuracy as m_i increases.

5.3. Estimation Accuracy. Figure 10 shows the cardinality estimation accuracy under three traffic traces for CSE, DCDS, SS, and our method called RSD. We can see that RSD has the minimum ARE of source cardinality estimation for each traffic trace from Figure 10(a). Similarly, RSD also has the minimum ARE of destination cardinality estimation for each traffic trace (Figure 10(b)). CSE, DCDS, SS, and RSD approximately estimate the cardinality of hosts using probabilistic methods, and their estimation accuracy depends on the memory utilization. RSD can simultaneously

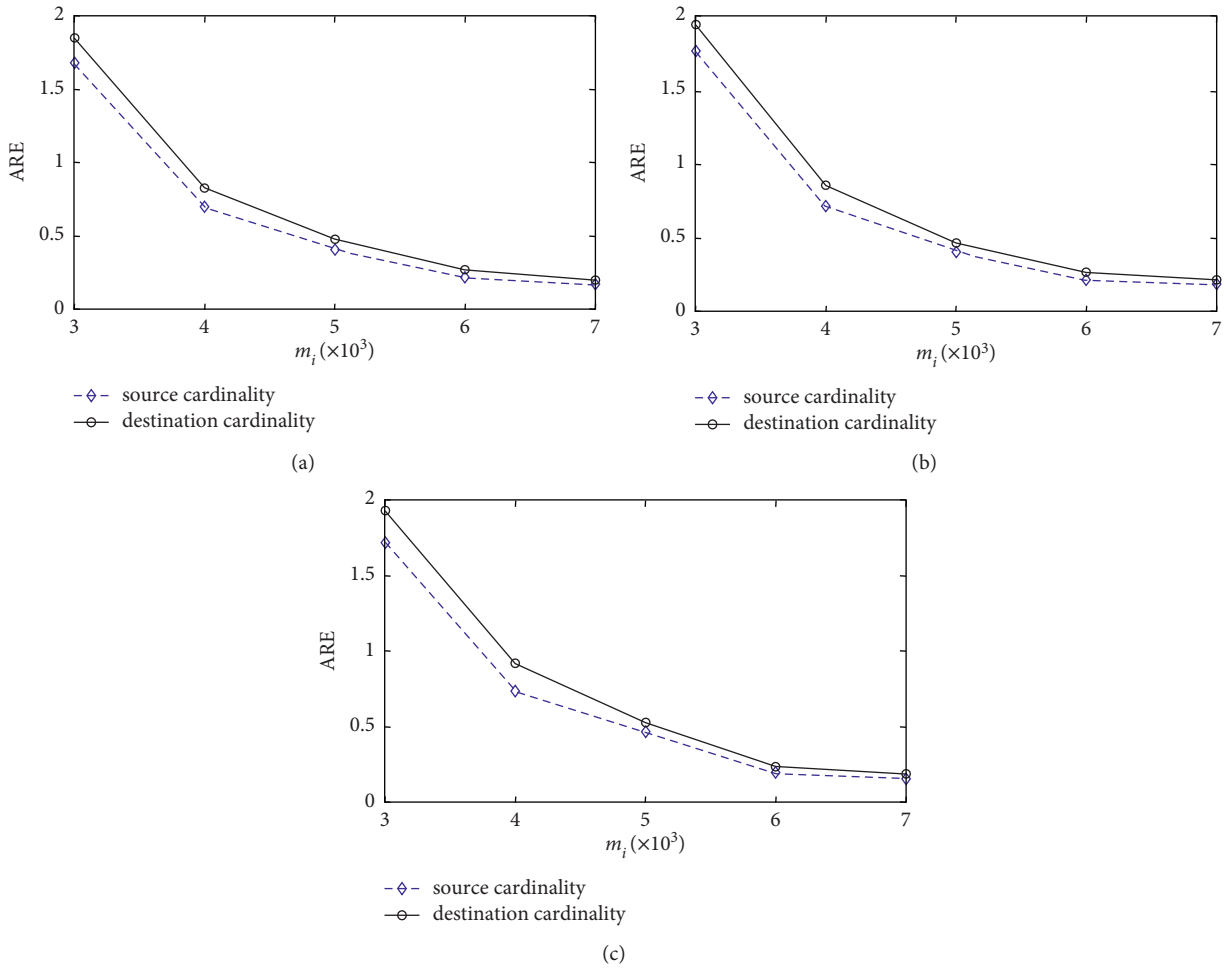

Figure 9: The influence of parameter m_i on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

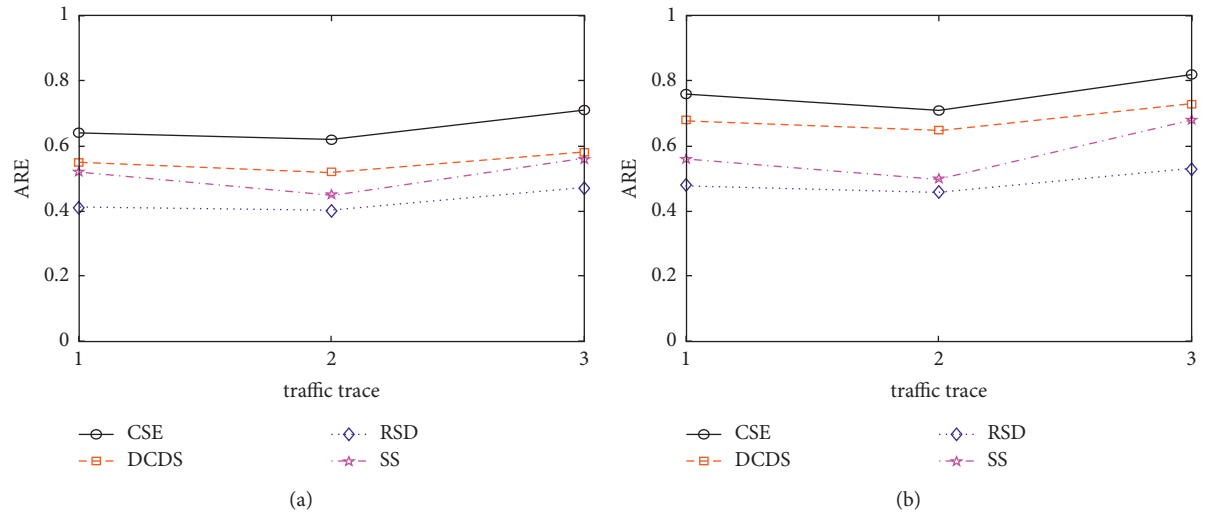

Figure 10: Cardinality estimation accuracy. (a) Source cardinality. (b) Destination cardinality.

and accurately estimate the cardinality of sources and destinations based on the same summary data structure generated by the controller in comparison with CSE, DCDS, and SS. However, the deviation between the estimated cardinality and the theoretical value still exists due to hash collisions.

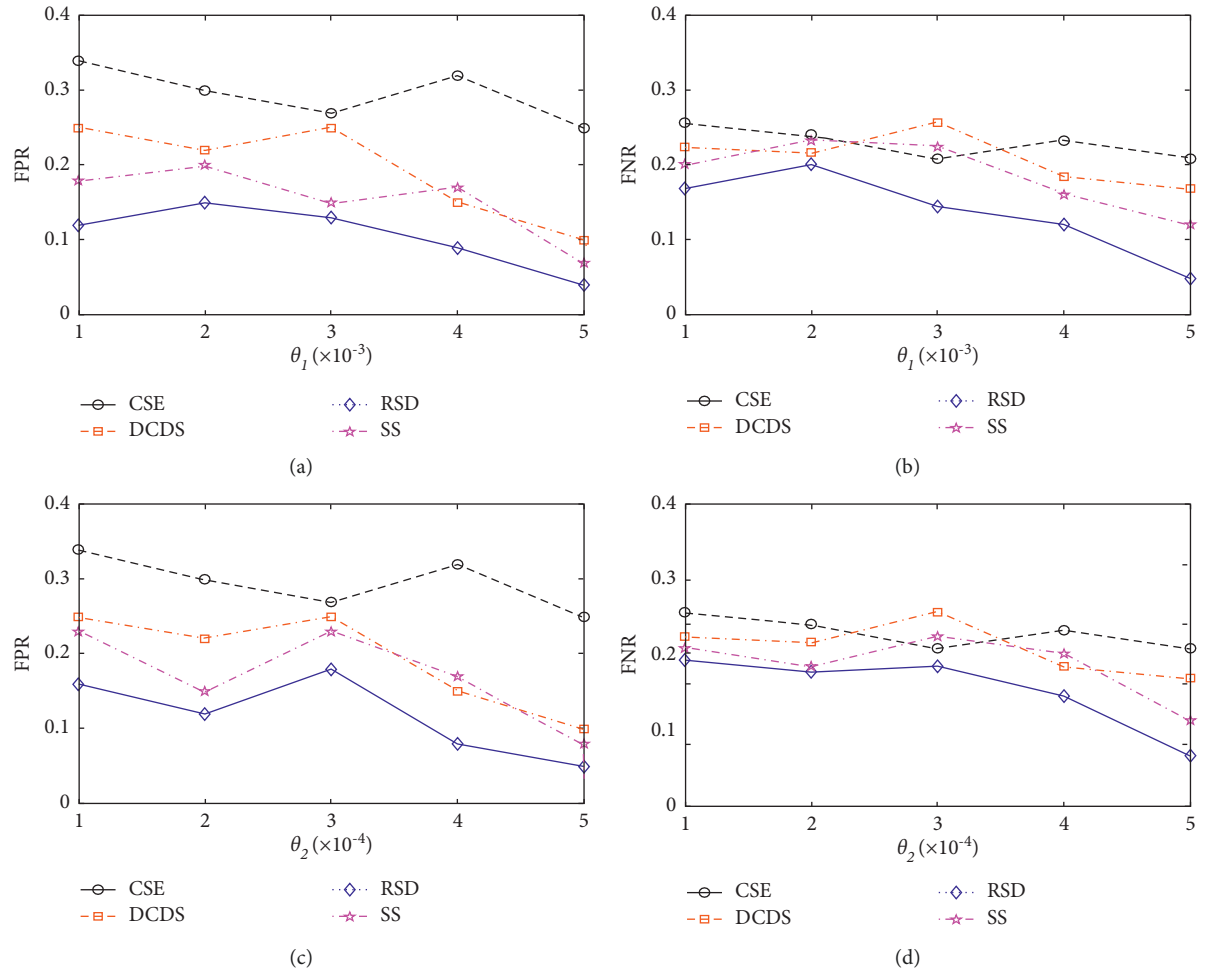

FIGURE 11: Detection precision under data1. (a) False positive rate of supersources. (b) False negative rate of supersources. (c) False positive rate of superdestinations. (d) False negative rate of superdestinations.

5.4. Detection Precision. We evaluate the performance of our algorithm called RSD compared with CSE, DCDS, and SS under different traffic traces. The reconstructed sources or destinations may not be true supersources or destinations by false combinations of abnormal rows or columns. Therefore, we use the false positive rate and false negative rate to evaluate the detection precision of four algorithms. The false positive rate (FPR for short) is the number of not corrected identified supernodes divided by the number of supernodes identified. The false negative rate (FNR) is the number of not identified supernodes divided by the number of true supernodes. The FPR and FNR are expressed as

$$\text{FPR} = \frac{|B - A|}{|B|}, \tag{32}$$

$$\text{FNR} = \frac{|A - B|}{|A|}, \tag{33}$$

where A is the set of true supernodes and B is the set of identified supernodes.

Figure 11 shows the detection precision of three algorithms under data1. The FPR and FNR of detecting supersources are shown in Figures 11(a) and 11(b). We can see that RSD has the lowest FPR and FNR for supersources in comparison with DCDS, CSE, and SS. The FPR changes from 0.04 to 0.15 and the FNR varies from 0.06 to 0.25 as the threshold θ_1 increases. The FPR and FNR of detecting superdestinations are shown in Figures 11(c) and 11(d). We can see that RSD has the lowest FPR and FNR for superdestinations compared to DCDS, CSE, and SS. The FPR changes from 0.05 to 0.18 and the FNR varies from 0.08 to 0.24 as the threshold θ_2 increases. Although there are some false positive and false negative due to the reconstruction of supersources and destinations using false combinations of abnormal rows and columns in two-dimensional bit arrays, RSD mitigates wrong reconstruction of supersources and destinations using the extra hash table that conducts the combinations of abnormal rows and columns. In general, RSD can simultaneously identify supersources and destinations based on their accurate cardinality estimation, and it outperforms DCDS, CSE, and SS in terms of FPR and FNR. Figure 12 shows the detection precision of three algorithms under data2. As shown in Figures 11(a)–11(d), we can obtain similar results.

5.5. Memory Cost. Figure 13 shows the memory cost of four algorithms under different traffic traces. The memory cost of our algorithm called RSD includes the

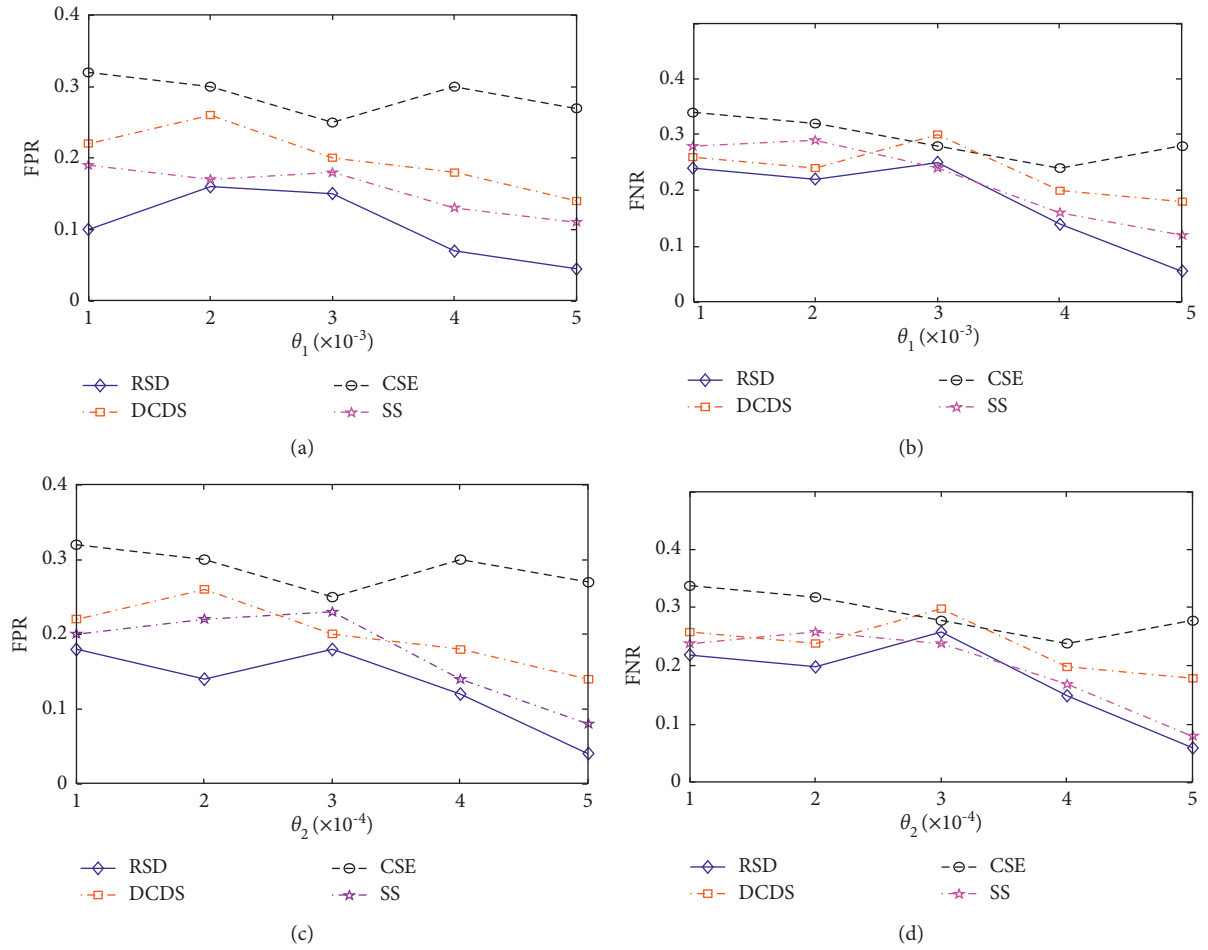

Figure 12: Detection precision under data2. (a) False positive rate of supersources. (b) False negative rate of supersources. (c) False positive rate of superdestinations. (d) False negative rate of superdestinations.

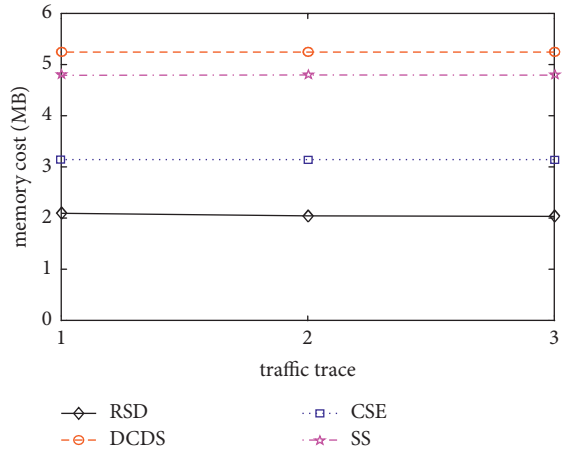

Figure 13: Memory cost under different traffic traces.

summary data structure, arrays, and hash table. The memory cost slightly increases when the extra bit arrays and hash table are created in the updating and reconstruction process. For simplicity, we treat the size of summary data structure as the memory cost. CSE, DCDS, and SS need to establish the data structure twice to simultaneously measure supersources and destinations, leading to high memory cost. Besides, DCDS uses the additional two-dimensional bit arrays to reduce the FPR caused by wrong combinations of abnormal rows and columns. CSE can only measure the cardinalities of supersources and destinations by constructing multiple virtual one-dimensional bit arrays and not reconstruct supersources and destinations. SS can measure the cardinalities of candidate supersources and destinations through building two-dimensional sketches, each bucket of which consists of a multiresolution bitmap, a label field, and a register. However, RSD can reduce the memory cost in supersource and destination detection by constructing row and column hash functions. In brief, the memory cost of RSD is superior to that of the three algorithms CSE, DCDS, and SS.

6. Conclusion

In this paper, we propose a novel method for detecting supernodes in the distributed monitoring systems. It constructs two-dimensional reversible summary data structures

to collect information associated with cardinalities according to the specific application requirements. At the end of each measurement period, the generated summary data structures are aggregated to produce the summary data structure with the same size. On the basis of the aggregated summary data structure, it estimates node cardinalities and reconstructs supersources and destinations. Compared to other algorithms, the proposed method can simultaneously measure two types of node cardinalities using the same summary data structures, detect supersources and destinations efficiently, and reconstruct labels of supersources and destinations with small computational complexity. We perform theoretical analysis and conduct extensive experiments on real traffic traces. The experimental results illustrate that our method has good estimation accuracy, detection precision, and memory cost. In the future, we will deploy our method and study superchanger detection problem in the practical distributed monitoring systems and study variants of supernode detection.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China, under Grant no. 61802274, Open Project Foundation of Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, China, under Grant no. K93-9-2017-01, and Scientific Research Foundation for Advanced Talents of Taizhou University, China, under Grant no. QD2016027.

References

[1] T. Yang, H. Zhang, J. Li et al., "HeavyKeeper: an accurate algorithm for finding Top-k elephant flows," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.

[2] J. Li, Z. Li, Y. Xu et al., "WavingSketch: an unbiased and generic sketch for finding top-k items in data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1574–1584, CA, USA, July 2020.

[3] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "FlowMon-DPDK: parsimonious per-flow software monitoring at line rate," in *Proceedings of the IEEE Network Traffic Measurement and Analysis Conference*, pp. 1–8, Vienna, Austria, June 2019.

[4] H. Wang, H. Xu, L. Huang, and Y. Zhai, "Fast and accurate traffic measurement with hierarchical filtering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2360–2374, 2020.

[5] Y. Du, H. Huang, Y.-E. Sun, S. Chen, and G. Gao, "Self-adaptive sampling for network traffic measurement," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–10, Vancouver, BC, Canada, May 2021.

[6] Y. Jing, R. Zhang, Y. Ma, Y. Zheng, L. Wu, and D. Zhang, "SuperSketch: a multi-dimensional reversible data structure for super host identification," *IEEE Transactions on Dependable and Secure Computing*1 page, 2021.

[7] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 164–176, Santa Clara, CA, USA, April 2017.

[8] Q. Huang and P. P. C. Lee, "A hybrid local and distributed sketching design for accurate and scalable heavy key detection in network data streams," *Computer Networks*, vol. 91, pp. 298–315, 2015.

[9] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *Proceedings of the IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, July 2020.

[10] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, Atlanta, GA, USA, July 2020.

[11] L. Tang, Q. Huang, and P. P. C. Lee, "MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 2026–2034, Paris, France, April 2019.

[12] Q. Xiao, Z. Tang, and S. Chen, "Universal online sketch for tracking heavy hitters and estimating moments of data streams," *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 974–983, Toronto, ON, Canada, July 2020.

[13] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1172–1185, 2020.

[14] J. Moraney and D. Raz, "On the practical detection of heavy hitter flows," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pp. 268–276, Bordeaux, France, May 2021.

[15] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 289–300, 2016.

[16] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, "Finding persistent items in distributed datasets," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 1–14, 2020.

[17] L. Chen, H. Dai, L. Meng, and J. Yu, "Finding needles in a hay stream: on persistent item lookup in data streams," *Computer Networks*, vol. 181, Article ID 107518, 2020.

[18] S. A. Singh and S. Tirthapura, "Monitoring persistent items in the union of distributed streams," *Journal of Parallel and Distributed Computing*, vol. 74, no. 11, pp. 3115–3127, 2014.

[19] C. Ma, S. Chen, Y. Zhang, Q. Xiao, and O. O. Odegbile, "Super spreader identification using geometric-min filter," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 299–312, 2022.

[20] P. Wang, X. Guan, T. Qin, and Q. Huang, "A data streaming method for monitoring host connection degrees of high-speed links," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1086–1098, 2011.

[21] W. Liu, W. Qu, J. Gong, and K. Li, "Detection of superpoints using a vector bloom filter," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 514–527, 2016.

[22] Y. Liu, W. Chen, and Y. Guan, "Identifying high-cardinality hosts from network-wide traffic measurements," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 547–558, 2016.

[23] G. Cheng and Y. Tang, "Line speed accurate superspreader identification using dynamic error compensation," *Computer Communications*, vol. 36, no. 13, pp. 1460–1470, 2013.

[24] J. Wang, W. Liu, L. Zheng, Z. Li, and Z. Liu, "A novel algorithm for detecting superpoints based on reversible virtual bitmaps," *Journal of Information Security and Applications*, vol. 49, Article ID 102403, 2019.

[25] R. Cohen and Y. Nezri, "Cardinality estimation in a virtualized network device using online machine learning," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2098–2110, 2019.

[26] H. Harmouch and F. Naumann, "Cardinality estimation," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 499–512, 2017.

[27] M. Roesch, "Snort: lightweight intrusion detection for networks," in *Proceedings of the USENIX International Conference on Systems Administration*, pp. 229–238, Seattle, WA, USA, June 1999.

[28] D. Plonka, "FlowScan: a network traffic flow reporting and visualization tool," in *Proceedings of the USENIX International Conference on Systems Administration*, pp. 305–317, New Orleans, LA, USA, December 2000.

[29] Y. E. Sun, H. Huang, and C. Ma el al., "Online spread estimation with non-duplicate sampling," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 2440–2448, Toronto, ON, Canada, July 2020.

[30] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual filter for non-duplicate sampling," *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, IEEE, in *Proceedings of the IEEE International Conference on Network Protocols*, pp. 1–11, November 2021.

[31] L. Zheng, D. Liu, W. Liu, Z. Liu, Z. Liu, and T. Wu, "A data streaming algorithm for detection of superpoints with small memory consumption," *IEEE Communications Letters*, vol. 21, no. 5, pp. 1067–1070, 2017.

[32] L. Wang, T. Yang, H. Wang et al., "Fine-grained probability counting for cardinality estimation of data streams," *World Wide Web*, vol. 22, no. 5, pp. 2065–2081, 2019.

[33] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, 2021.

[34] Y. Zhou, Y. Zhou, S. Chen, and Y. Youlin Zhang, "Per-flow counting for big network data stream over sliding windows," *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, IEEE, in *Proceedings of the IEEE/ACM International Symposium on Quality of Service*, pp. 1–10, June 2017.

[35] J. Qi, W. Li, T. Yang, D. Li, and H. Li, "Cuckoo counter: a novel framework for accurate per-flow frequency estimation in network measurement," *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, IEEE, in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1–7, September 2019.

[36] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: accurate per-flow measurement for big streaming data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2650–2662, 2019.

[37] Y. Zhou, P. Liu, H. Jin, T. Yang, S. Dang, and X. Li, "One memory access sketch: a more accurate and faster sketch for per-flow measurement," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, in *Proceedings of the IEEE Global Communications Conference*, pp. 1–6, Singapore, 4 December 2022.

[38] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, Honolulu, HI, USA, 23 April 2006.

[39] R. Shahout, R. Friedman, and D. Adas, "CELL: counter estimation for per-flow traffic in streams and sliding windows," in *Proceedings of the IEEE International Conference on Network Protocols*, pp. 1–12, Dallas, TX, USA, 13 Oct. 2020.

[40] Q. Liu, H. Dai, A. X. Liu, Q. Li, X. Wang, and J. Zheng, "Cache assisted randomized sharing counters in network measurement," in *Proceedings of the ACM International Conference on Parallel Processing*, pp. 1–10, Eugene, OR, USA, August 2018.

[41] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.

[42] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, 2011.

[43] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: improving the performance of cardinality estimation of large data streams," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, IEEE, Atlanta, GA, USA, 1 May 2017.

[44] P. Wang, P. Jia, J. Tao, and X. Guan, "Detecting a variety of long-term stealthy user behaviors on high speed links," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1912–1925, 2019.

[45] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, 5 pages, 2008.

[46] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the International Conference on Database Theory*, pp. 398–412, Edinburgh, UK, January 1999.

[47] F. Wang and L. Gao, "Simple and efficient identification of heavy hitters based on bitcount," in *Proceedings of the IEEE International Conference on High Performance Switching and Routing*, pp. 1–6, Xi'an, China, 6 June 2022.

[48] H. Huang, Y. E. Sun, C. Ma et al., "Spread estimation with non-duplicate sampling in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2073–2086, 2021.

[49] L. Tang, Q. Huang, and P. P. C. Lee, "SpreadSketch: toward invertible and network-wide detection of superspreaders," *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1608–1617, Toronto, ON, Canada, 6 July 2020.

[50] T. Yang, J. Jiang, P. Liu et al., "Elastic sketch: adaptive and fast network-wide measurements," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 561–575, New York, NY, USA, August 2018.

[51] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide

measurements," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2386–2398, 2021.

[52] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 1, 67 pages, 2017.

[53] H. Huang, Y. E. Sun, C. Ma et al., "An efficient k-persistent spread estimator for traffic measurement in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1463–1476, 2020.

[54] MAWI Dataset, https://mawi.wide.ad.jp/mawi/, 2020.