algorithms

MDPI

Article

Rough Estimator Based Asynchronous Distributed Super Points Detection on High Speed Network Edge

Jie Xu [1,*,†] and Wei Ding [2,†]

[1] Jiangsu Police Institute, Nanjing 210031, China
[2] School of Cyber Science and Engineering, Southeast University, Nanjing 211102, China; wding@njnet.edu.cn
* Correspondence: xujieip@163.com
† These authors contributed equally to this work.

Abstract: Super points detection plays an important role in network research and application. With the increase of network scale, distributed super points detection has become a hot research topic. The key point of super points detection in a multi-node distributed environment is how to reduce communication overhead. Therefore, this paper proposes a three-stage communication algorithm to detect super points in a distributed environment, Rough Estimator based Asynchronous Distributed super points detection algorithm (READ). READ uses a lightweight estimator, the Rough Estimator (RE), which is fast in computation and takes less memory to generate candidate super points. Meanwhile, the famous Linear Estimator (LE) is applied to accurately estimate the cardinality of each candidate super point, so as to detect the super point correctly. In READ, each node scans IP address pairs asynchronously. When reaching the time window boundary, READ starts three-stage communication to detect the super point. This paper proves that the accuracy of READ in a distributed environment is no less than that in the single-node environment. Four groups of 10 Gb/s and 40 Gb/s real-world high-speed network traffic are used to test READ. The experimental results show that READ not only has high accuracy in a distributed environment, but also has less than 5% of communication burden compared with existing algorithms.

Keywords: super points detection; distributed computing; network measurement; network security

Citation: Xu, J.; Ding, W. Rough Estimator Based Asynchronous Distributed Super Points Detection on High Speed Network Edge. *Algorithms* **2021**, *14*, 277. https://doi.org/10.3390/a14100277

Academic Editor: Charalampos Konstantopoulos and Grammati Pantziou

Received: 1 September 2021
Accepted: 20 September 2021
Published: 25 September 2021

1. Introduction

The Internet is one of the most important infrastructures of the modern information society. With the rapid development of China's economy, the bandwidth of the core network is increasing year by year. According to the latest statistics of China Internet Information Center (CNNIC), as of December 2018, China's international export bandwidth has reached 8,946,570 Mbps, with an annual growth rate of 22.2% [1]. It is a worldwide problem to manage such a large-scale network effectively and ensure its safe operation.

In the face of a complex network environment, the monitoring and protection of the backbone network is the most important and basic step [2]. Internet management under the condition of large data-level network traffic is a hot research subject, which can be carried out from different aspects at the industrial and academic levels. To pay more attention to some core hosts in the network is a way to improve the efficiency of network management [3].

The super point in the Internet is such a kind of core host [4]. It is generally believed that a super point refers to a host that communicates with many other hosts. Super points play important roles in the network, such as servers, proxies, scanners [5], hosts attacked by DDoS, etc. The detection and measurement of super points are important to network security and network management [6].

With the increase of network size, large-scale networks usually contain multiple border entries and exits. How to detect the super point from multiple nodes is a new requirement for super points detection. Some existing algorithms, such as Double Connection Degree

Sketch algorithm (DCDS) [7], Vector Bloom Filter Algorithm (VBFA) [8] and Compact Spread Estimator (CSE) [9] and so on, can realize distributed super points detection by adding data merging process. However, in the distributed environment, DCDS, VBFA, CSE must send all the whole used memory, which is more than 300MB for a 10Gb/s network, to the main server. When detecting the super point, such a large data transmission between the sub-node and the global server will cause the peak traffic of network communication and increase the communication delay. How to reduce the communication overhead in a distributed environment is a difficult problem in the research of distributed super points detection.

Super points account for only a small portion of all hosts. In theory, only the data related to the super point should be sent to the global server to complete the super points detection. Based on this idea, a distributed super points detection algorithm, asynchronous distributed algorithm based on rough estimator (READ), is proposed in this paper. READ uses a lightweight rough estimator (RE) to generate candidate super points. Because RE takes up less memory, each sub-node only needs to send a small amount of data to the global server to generate candidate super points. READ not only reduces the detection error rate, but also reduces the communication overhead by transferring data related to candidate super points to the global server.

Part of this paper has been published at the conference of Algorithms and Architectures for Parallel Processing 2018 [10]. This paper extends from the aspects of algorithm introduction, theoretical analysis, and experimental demonstration.The main contributions of this paper are as follows:

- A method of generating candidate super points in a distributed environment using lightweight estimators is proposed.
- A distributed super points detection algorithm READ with low communication overhead is proposed.
- Prove theoretically that READ has lower error rate in a distributed environment.
- Using the real-world high-speed network traffic to evaluate the performance of READ.

Section 2 introduces the rough estimator and the linear estimator for estimating the host's cardinality, as well as the existing algorithms for super points detection. Section 3 discusses the model and difficulty of distributed super points detection. Section 4 introduces the operation principle of READ, and how READ works. Section 5 introduces how to modify READ to work under a sliding time window. Section 6 shows the experiment of READ with 10 Gb/s and 40 Gb/s real world network traffic, and analyzes the detection accuracy of READ in a distributed environment and the communication overhead between sub-nodes and the global server. Section 8 summarizes READ.

2. Related Work

Super points detection is a hotspot in the field of network research and management. For the sake of narrative convenience, this section first gives relevant definitions.

2.1. Related Definitions

Information security is becoming more and more important to people's life [11]. How to discover abnormal traffic or hosts from a high-speed network is one of the important topics in the field of security research. Super points detection is one of the important methods for locating anomaly hosts [12]. All of the super points detection algorithms are based on network traffic and belong to passive network measurement. The original data used in the algorithm is the IP address collected from the network. For network managers, the measuring place is usually located at the boundary of the managed network, as shown in Figure 1. Observation node is a server beside a router, from which the packets between two networks could be collected and inspected. The host in $\mathbb{A}$ communicates with those hosts in $\mathbb{B}$ through the boundary router. IP address pairs such as $< \mathtt{a}, \mathtt{b} >$ can be extracted from each packet passing through the border router, where $\mathtt{a} \in \mathbb{A}$, $\mathtt{b} \in \mathbb{B}$. For the host $\mathtt{a}$ in $\mathbb{A}$, its cardinality is defined as follows:

Definition 1 (Opposite host set/cardinality). *In time window $\mathscr{T}$, for a host $\mathbb{a} \in \mathbb{A}$, the set of all hosts in $\mathbb{B}$ that communicating with it is called the opposite host set of $\mathbb{a}$, and is denoted as $\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}$. The size of $\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}$ is called the cardinality of $\mathbb{a}$, which is denoted as $|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|$.*

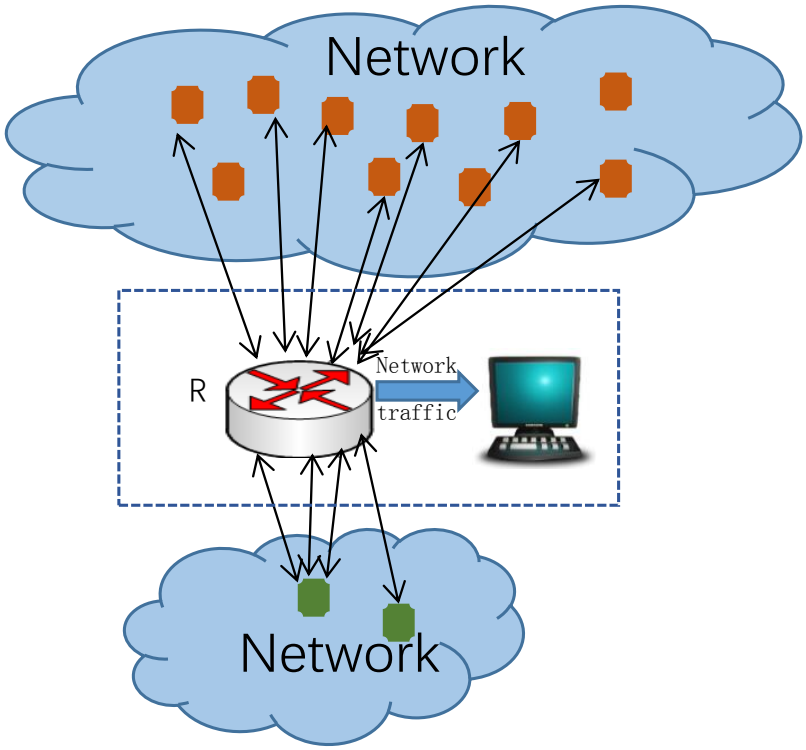

Figure 1. The observation node on network boarder.

The cardinality is one of the important network attribute[13], and it is the criteria to judge if the host is a super point.

Definition 2 (Super point). *In the time window $\mathscr{T}$, the host whose cardinality exceeds the specified threshold θ is called a super point.*

In this paper, without losing generality, it is assumed that the super points detection is only for $\mathbb{A}$. Threshold θ is set by the users according to different situations, such as detecting DDoS attacks, locating servers and so on.

Cardinality estimation is the basis of super points detection. The next section will introduce the commonly used algorithm for cardinality estimating in super points detection.

2.2. Cardinality Estimation

Cardinality is an important attribute in network research [14]. At the same time, the calculation of cardinality is also the basis of super points detection [15]. Therefore, this sub section introduces the algorithm of host's cardinality estimating [16].

There are many cardinality estimating algorithms, such as Probabilistic Counting Statistic Algorithm (PCSA) [17], HyperLogLog algorithm [18], Linear Estimator (LE) algorithm [19] and so on. LE algorithm is widely used in super points detection because of its high accuracy and simple operation.

Let $\mathbb{C}$ denote a set of bits and $|\mathbb{C}|$ denote the number of bits in $\mathbb{C}$. LE uses $\mathbb{C}$ to record and estimate the opposite hosts of $\mathbb{a}$. Each bit in $\mathbb{C}$ is initially set to zero. For any opposite

host $\mathbb{b}$, LE maps it to a bit in $\mathbb{C}$ by using the hash function $\mathbb{h}^{LE}(\mathbb{b})$ and sets the bit to 1. At the end of time window $\mathscr{T}$, LE uses the following formula to estimate $|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|$.

$$|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|' = -|\mathbb{C}| * log(n_0/(|\mathbb{C}|)) \tag{1}$$

where n_0 denotes the number of bits in C with value of 0. The estimation error of LE is related to $|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|$ and the number of counter $|\mathbb{C}|$. Define the ratio of $|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|$ to $|\mathbb{C}|$ as a load factor, marked $\mathfrak{L}$. The estimated standard deviation of LE is $\sqrt{\frac{(e^{\mathfrak{L}}-\mathfrak{L}-1)}{\mathbb{C}}}$.

When $|\mathbb{S}^{\mathbb{B}}_{\mathbb{a},\mathscr{T}}|$ is determined, the larger $|\mathbb{C}|$ is, the higher the estimation accuracy of LE is. However, the larger $|\mathbb{C}|$, the more memory space LE occupies, and the longer time it takes to estimate the cardinality.

In order to compensate for the deficiency of LE, Jie et al. [20] proposed a lightweight rough estimator (RE). RE only takes eight bits to determine whether $\mathbb{a}$ is a candidate super point. At initialization, RE sets all eight bits to 0. For each opposite host $\mathbb{b}$ of $\mathbb{a}$, RE maps $\mathbb{b}$ to a random integer $\tilde{\mathbb{b}}$ between 0 and $2^{32}-1$ using hash function $\mathbb{h}^{rand}(\mathbb{b})$, and then compares the lowest significant bit of $\tilde{\mathbb{b}}$ with a real number τ. The lowest significant bit is the position of the first bit "1" starting from the right. For example, the binary formatter of integer 200 is "11001000", its lowest significant bit is 3. Let $\mathscr{R}^0(x)$ denote the lowest significant bit of integer x. τ is used to determine whether update a bit. The definition of τ is as follows.

$$\tau = log2(\theta/8) \tag{2}$$

If $\mathscr{R}^0(\tilde{\mathbb{b}}) \geq \tau$, RE maps $\mathbb{b}$ to one of eight bits using a hash function and sets the bit to 1. Denote this hash function as $\mathbb{h}^{RE}(\mathbb{b})$. When the number of bits with a value of 1 is greater than or equal to 3, RE determines $\mathbb{b}$ as a candidate super point. As a lightweight estimator, RE can quickly determine candidate super point, but it cannot accurately estimate the cardinality. Jie et al. [21] used RE as a preliminary screening tool to reduce the range of candidate super points, and combined with LE to realize real-time detection of super points under a sliding time window. A detailed analysis of RE can be found in [22].

2.3. Super Points Detection

From the introduction in the previous sub section, LE and RE can estimate the cardinality of a host and determine whether a host is a candidate super point. However, there are a large number of active IPs [23] in the actual network. At the beginning of the time window, it is not known which IP will become a super point. The task of the super points detection algorithm is to detect the super points from these IPs based on the cardinality estimation algorithm. In this paper, the memory that used to record the opposite hosts' information is called a master data structure.

A simple and straightforward method of super points detection is to record each host $\mathbb{a}$ and its opposite IP. However, this is unrealistic, because there are many IP addresses in high-speed networks. Accurately recording each IP and its opposite host not only requires a lot of memory, but also a lot of memory access times [24]. Therefore, the estimation-based super points detection algorithms using fixed amount of memory have attracted wide attention, and a large number of super points detection algorithms have emerged, such as CBF [12], DCDS [7], VBFA [8] and CSE [9].

CBF [12] is a super points detection algorithm based on the principle of Bloom filter. It uses Bloom filter to remove duplicate IP address pairs, and uses a data structure derived from Bloom filter, called Counting Bloom filter, to record opposite IP information. The algorithm uses Bloom filter to avoid multiple updates of the master data structure by the same IP address pair, and improves the speed of the algorithm. When updating the counting Bloom filter, only increment some counters with 1, and no other complicated calculation is needed. Since each counter can be used by multiple hosts, the memory usage of the algorithm is low. Although Bloom filter can avoid multiple updates of CBF to an IP address pair, it may also cause omissions of some IP address pairs. In a distributed

environment, an IP address pair will appear on different nodes, which will be updated by different nodes many times. Therefore, CBF cannot be applied to distributed environment.

DCDS [7], VBFA [8] and CSE [9] all use LE to estimate host's cardinality. DCDS [7] uses China Remainder Theorem (CRT) [25] to restore candidate super point. However, when mapping $\mathtt{a}$ to LE, DCDS needs to use CRT principle, which takes up more computing time and is not conducive to the improvement of algorithm speed. VBFA does not use computationally complex CRT to recover candidate super points, but maps $\mathtt{a}$ to different LE according to the principle of Bloom filter [26]. The length of LE array used to recover candidate super points in VBFA is fixed. As the number of host increases, each LE is used to estimate too many hosts' cardinalities. At this time, the number of hot LE (whose cardinality is bigger than the threshold) in LE array increases correspondingly. The number of hot LEs that need to be tested also increases, which increases the time to recover candidate super points. CSE uses virtual LE to estimate the number of counterparts. CSE assigns a virtual LE to each $\mathtt{a}$. Each bit virtual LE associates with a physical bit in the bit pool. CSE achieves bit-level sharing and makes more efficient use of memory. Each $\mathtt{a}$ associates with only a virtual estimator, so only one physical bit needs to be updated when scanning each IP address pair, and memory access times are less than DCDS and VBFA. CSE cannot generate candidate super points after scanning all IP address pairs in a time window like DCDS and VBFA. Therefore, CSE saves all hosts in $\mathbb{A}$ as candidate super points, when scanning IP address pairs. It increases the number of candidate super points and the time used to estimate the cardinalities of candidate super points.

DCDS, VBFA and CSE can run in a distributed environment. In a distributed environment, DCDS and VBFA collect LE from all nodes, and merge these LE sets according to "bit or" mode; CSE collects bit pools from all nodes, and merges these bit pools according to "bit or" mode. Then, the super points are detected according to the unioned LE set or bit pool. Although DCDS, VBFA and CSE can run in a distributed environment, they need to collect all LE or bit pools from each distributed node, which leads to low communication efficiency. This paper presents an algorithm that can realize distributed super points detection by collecting only fraction of LE sets, which reduces the communication in a distributed environment.

2.4. Notations and Symbols

To facilitate reading, Table 1 lists some commonly used symbols and abbreviations in this article. In Table 1, RE cube, RE array, LE array are data structures used in the novel algorithm, and they will be described in detail in Section 4.

Table 1. Notations and symbols used.

Notation	Definition
$\mathbb{A}$	The network from which to detect super points.
$\mathbb{B}$	The network communicating with $\mathbb{A}$ through edge routers.
$\mathtt{a}$ or $\mathtt{b}$	An IP address in $\mathbb{A}$ or $\mathbb{B}$.
$\mathscr{T}$	A time window.
$\mathbb{S}^{\mathbb{B}}_{\mathtt{a},\mathscr{T}}$	Set of opposite hosts of $\mathtt{a}$ in $\mathscr{T}$.
$\mathfrak{n}$	The number of distributed observation nodes.
$\mathscr{O}_{\mathfrak{l}}$	The $\mathfrak{l}$-th observation node.
$\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}}$	The stream of IP pair observed on $\mathscr{O}_{\mathfrak{l}}$ in time window $\mathscr{T}$.
$\mathbb{R}^{\mathfrak{l}}$	A RE cube in the $\mathfrak{l}$-th observation node.
r	The number of right bits in $\mathtt{a}$ used to locate a RE array in RE cube.
$\mathscr{L}_{\mathtt{a}}$	The left $(32-r)$ bits of $\mathtt{a}$.
u	The number of row in a RE array.
v_i	The number of bits in $\mathscr{L}_{\mathtt{a}}$ which is used to locate a RE in the i-th row of a RE array.
$\mathbb{L}^{\mathfrak{l}}$	A LE array in the $\mathfrak{l}$-th observation node.
$\hat{u}$	The number of row of a LE array.
$\hat{v}$	The number of column of a LE array.

3. Distributed Super Points Detection MODEL and Difficulty

A network connected to the Internet may have multiple border routers, as shown in Figure 2. For example, a campus network access to multiple Internet Service Provider(ISP). In Figure 2, there are three host in the bottom network. Each host can communicate with the host in the other network through different routers. When detecting super points, the opposite host set must be collected from all routers. For example, the middle host in the bottom network communicate with more than six opposite hosts through all routers. When the cardinality threshold is 5, the middle host in the bottom network is a super point. Assuming that there is an observation node at each border router. Traffic can be observed and analyzed independently on each node. This section will discuss the algorithm of super points detection in a distributed environment.

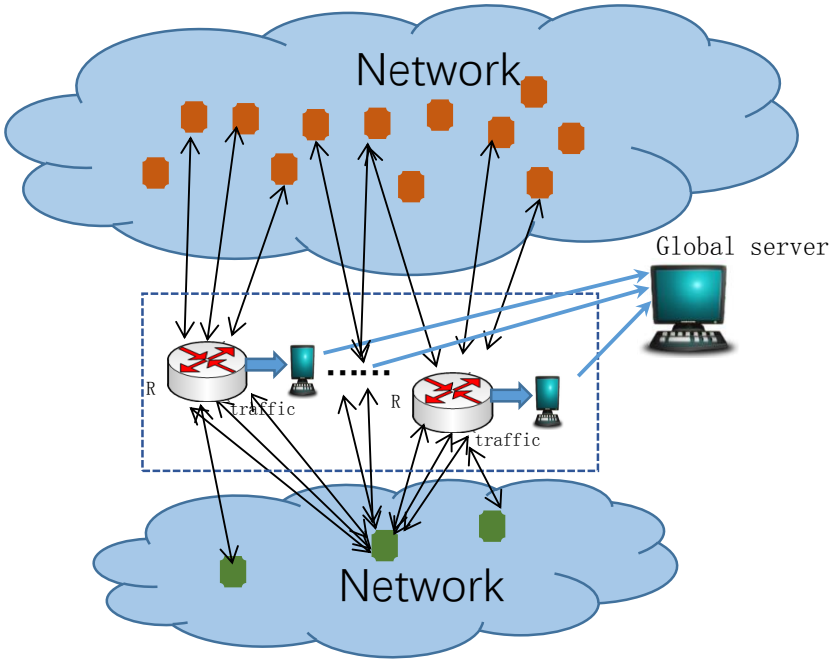

Figure 2. Super points detection in a distributed environment.

3.1. Detection Model

For a host a in the network, it may interact with different opposite hosts through different border routers. At this time, only part of the traffic of a can be observed at each observation node. Assuming that the host a communicates with other networks in the Internet through n border routers, only part of the traffic of a is forwarded on each border router. At this time, the cardinality of a observed at each border router may be less than the threshold, but the cardinality of a observed from all observation nodes is larger than the threshold, which will lead to the omission of super points. Therefore, it is a meaningful work to detect the super point in a distributed environment.

In the distributed environment, the global server collects data from all observation nodes and performs super points detection. The research of super points detection in a distributed environment is to study which data the global server collects from the observation nodes and how to detect the global super points on the global server.

3.2. Requirements and Difficulties

In order to find all super points in a distributed environment, it is necessary to detect them globally. A simple method is to send the IP address pairs extracted from each observation node to a global server that processes all data, and then detect the super point on the global server. This method needs to transfer a large amount of data between the global server and observation nodes. Therefore, the method of sending all IP addresses

to the global server and detecting the super point on the global server cannot process the high-speed network data in real time because of the long communication time.

Another method of super points detection in a distributed environment is to run super points detection algorithms, such as DCDS, VBFA and CSE, at each observation node and then send only the master data structure to the global server for super points detection. Compared with the method of transferring all IP addresses to the global server, the method of transferring only the master data structure to the global server reduces the communication overhead between observation nodes and the global server.

However, when using this method, all observation nodes need to transmit the master data structure to the global server. When the number of observation nodes increases, the total amount of data transferred between all observation nodes and the global server will also increase. Moreover, the size of the master data structure is related to the error rate of the algorithm: the larger the master data structure, the lower the error rate of the algorithm. Therefore, the communication overhead between the observation node and the master node cannot be reduced by reducing the size of the master data structure. In addition, the transmission of all master data structures will generate a large amount of burst traffic at the end of the time window, which will increase the network burden.

How to avoid sending all master data structures to the global server and reduce the communication between observation nodes and the global server is a difficult problem in a distributed environment.

3.3. Solution of This Paper

If only part of the cardinality estimation structure at the observation node is sufficient to detect the global super point, then there is no need to transfer all of them between the observation node and the global server, which can further reduce the communication overhead. Based on this idea, this paper proposes a low communication cost distributed super points detection algorithm: Rough Estimator based Asynchronous Distributing Algorithm (READ).

In a distributed environment, it is necessary to recover the global candidate super points at the end of the time window according to the information recorded at all observation nodes. DCDS and VBFA have the function of recovering candidate super points. However, DCDS and VBFA have to use LE to recover candidate super points. Although LE has a high accuracy, it also occupies a high amount of memory, resulting in a large amount of communication between observation nodes and the global server.

RE not only runs fast, but also occupies less memory. If RE is used to generate candidate super points, a small amount of memory can be used to generate global candidate super points. The global server collects LE related to candidate super points from all observation nodes for estimating the cardinalities of candidate super points, and then completes super points detection without transmitting all cardinality estimation structure. The next section will describe how READ works.

4. RE Based Distributed Super Points Detection Algorithm READ

This section will introduce the novel low communication overhead distributed super points detection algorithm Rough Estimator based Asynchronous Distributed super points detection algorithm (READ).

4.1. Principle of READ

READ uses a data structure that can recover candidate super points to achieve distributed super points detection. It uses RE to recover candidate super points and LE to estimate cardinality of each candidate super point. Therefore, the master data structure of READ includes two parts: RE set and LE set. Scanning IP address pairs and estimating cardinalities are operations on RE and LE sets. REDA contains three main steps:

- Scan IP pair on each observation node. Each observation node scans each IP address pair passing through it and updates the RE and LE sets on it.

- Generate candidate super points in global server. The global server collects RE sets from all observation nodes, merges these RE sets, and generates candidate super points using the merged RE sets.
- Estimate cardinalities and filter super points. After the candidate super points are obtained, the global server collects LE related to each candidate super point from all observation nodes, and estimates the cardinalities of candidate super points based on these LE.

According to the above analysis, in READ, the communication between observation nodes and the global server is divided into three stages:

- Each observation node sends RE set to global server;
- The global server distributes candidate super points to each observation node;
- Each observation node sends LE of every candidate super point to the global server;

For READ, the sum of the communication in the three stages above is the total communication between an observation node and the global server in a time window. The number of LEs sent by observation nodes to the global server equals to the number of candidate super points. Since the number of candidate super points is less than the number of LE in the master data structure, the amount of data sent by each observation node to the global server is less than the size of LE set.

4.2. Scanning IP Pair in a Distributed Environment

Distributed scanning IP address pairs is to scan the IP address pairs collected at each observation node. Let $\mathscr{O}_{\mathfrak{l}}$ denote the $\mathfrak{l}$-th observation node and $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l},d}$enote all IP address pairs in time window $\mathscr{T}$ on $\mathscr{O}_{\mathfrak{l}}$. READ uses RE estimator and LE estimator to record IP information. Each observation node has the same cardinality estimation structure: the same number of RE and LE, and the same number of counters in RE and LE. The basic operation of $\mathscr{O}_{\mathfrak{l}}$ when scanning IP address pairs is to update RE and LE.

READ uses RE to generate global candidate super points, and LE to estimate the cardinality of each global candidate super point. In a distributed environment, because only part of the network traffic can be observed at each observation node, it is impossible to determine whether a host is a global candidate super point according to RE when scanning IP address pairs. In a distributed environment, the algorithm of super points detection must be able to recover the global candidate super points directly, such as DCDS and VBFA.

In order to recover candidate super points, READ adopts a new data structure, Rough Estimator Cube (REC). REC is a three-dimensional data structure composed of RE, as shown in Figure 3. Inspired by VBFA, READ restores candidate super points by concatenating sub bits of RE indexes in REC.

The basic element of REC is RE. Several RE constitutes a one-dimensional RE vector (REV); the set of REV constitutes a two-dimensional RE array (RE Array, REA). The three-dimensional REC can be regarded as a set of REA, which contains 2^r REA and r is a positive integer less than 32. Each REA of REC has the same structure, that is, the REA contains the same number of REV, and the associating REV contains the same number of RE. Let u denote the number of REV contained in REA and 2^{v_i} denote the number of RE contained in the ith REV. Three indexes can be used to locate a RE in REC accurately.

All observation nodes have their own REC, and the structure of REC at different observation nodes is the same, that is, the r, u , 2^{v_i} of REC at different observation nodes are the same. When the IP address pair is scanned at the observation node, the REC at the observation node will be updated. Let $\mathbb{R}^{\mathfrak{l}}$ denote the REC on the observation node $\mathscr{O}_{\mathfrak{l}}$, $\mathbb{R}^{\mathfrak{l}}_{(i,j,k)}$ denote the j-th RE of the i-th REV on the k-th REA, where k is an integer between 0 and $2^r - 1$, i is an integer between 0 and $u - 1$, and j is an integer between 0 and $2^{v_i} - 1$. In time window $\mathscr{T}$, for each IP address pair $< \mathbb{a}, \mathbb{b} >$ of $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}_{,}}$ READ selects u RE from $\mathbb{R}^{\mathfrak{l}}$ according to $\mathbb{a}$, and updates u RE with $\mathbb{b}$. How to map $\mathbb{a}$ to u RE in REC determines whether READ can recover global candidate super points from REC.

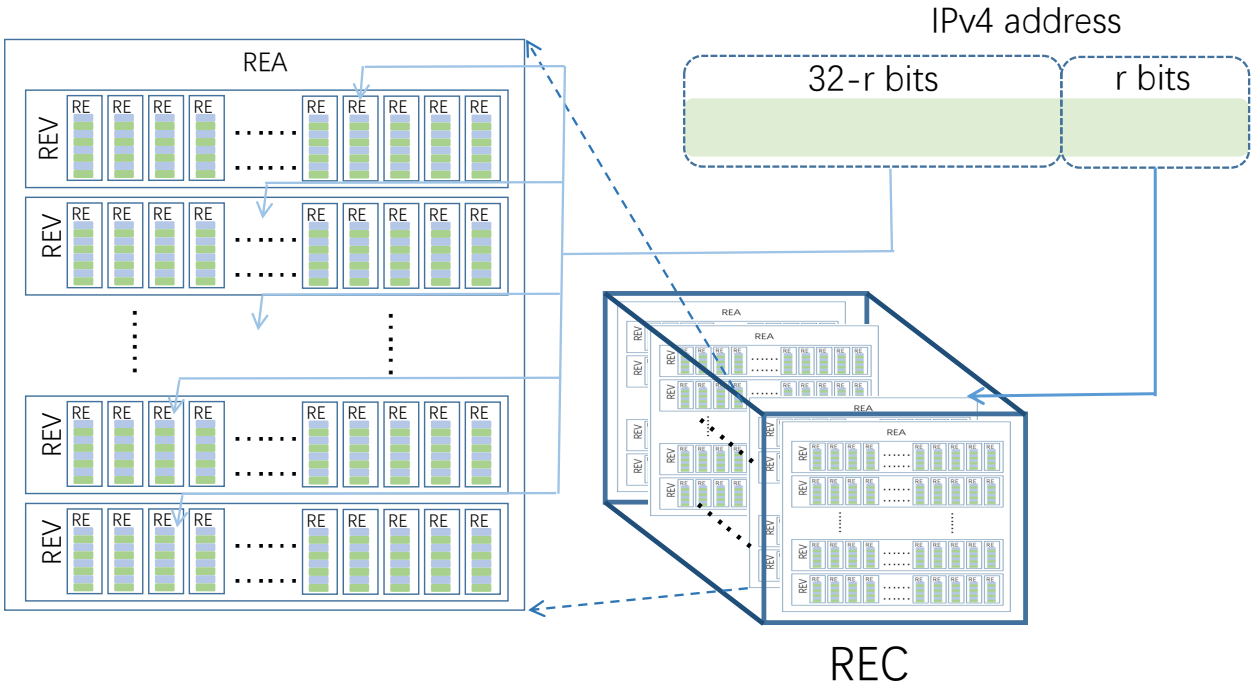

Figure 3. Structure of RE cube.

The u RE associating with $\mathbb{a}$ are located in the same REA. READ divides $\mathbb{A}$ into two parts: the first part is r bits on the right (Right Part, RP), and the second part is 32-r bits on the left (Left Part, LP).

READ selects a REA in the REC based on the IP of $\mathbb{a}$. REC has 2^r REA, so the RP of $\mathbb{a}$ can determine only one REA in the REC. READ divides $\mathbb{A}$ into 2^r subsets according to r bits on the right side of the IP address. Each subset of $\mathbb{A}$ associates with a REA in the REC. During the operation of the algorithm, the number of RE in the REC is fixed, and each RE is used to record opposite hosts of multiple $\mathbb{a}$. When $\mathbb{A}$ contains many IP addresses, by increasing r, the number of hosts sharing the same RE can be reduced.

The LP of $\mathbb{a}$ is used to select u RE in REA, i.e., one RE from each REV. Let $\mathscr{I}_{\mathbb{a}}^{i}$ denote the index of RE in the i-th REV, $0 \leq \mathscr{I}_{\mathbb{a}}^{i} \leq 2^{v_i} - 1$. $\mathscr{I}_{\mathbb{a}}^{i}$ is an integer containing v_i bits. Let $\mathscr{I}_{\mathbb{a}}^{i}$ [j] denote the j-th bit in $\mathscr{I}_{\mathbb{a}}^{i}$, $0 \leq j \leq v_i - 1$. READ selects v_i bits from the LP of $\mathbb{a}$ as the value of $\mathscr{I}_{\mathbb{a}}^{i}$. Let $\mathscr{L}_{\mathbb{a}}$ denote the LP of $\mathbb{a}$, $\mathscr{L}_{\mathbb{a}}$[i] denote the i-th bit of $\mathscr{L}_{\mathbb{a}}$, $0 \leq i \leq 32-\mathrm{r}-1$. Each bit in $\mathscr{I}_{\mathbb{a}}^{i}$ associates with a bit in $\mathscr{L}_{\mathbb{a}}$, as shown in Figure 4.

When selecting bits from $\mathscr{L}_{\mathbb{a}}$ as $\mathscr{I}_{\mathbb{a}}^{i}$, READ first determines which bit in $\mathscr{L}_{\mathbb{a}}$ is $\mathscr{I}_{\mathbb{a}}^{i}$[0], and then calculates the other bits in $\mathscr{I}_{\mathbb{a}}^{i}$. Let $\mathfrak{b}_i$ denote the index of the 0th bit of $\mathscr{I}_{\mathbb{a}}^{i}$ in $\mathscr{L}_{\mathbb{a}}$, i.e., $\mathscr{I}_{\mathbb{a}}^{i}[0]$=$\mathscr{L}_{\mathbb{a}}[\mathfrak{b}_i]$. Each bit of $\mathscr{I}_{\mathbb{a}}^{i}$ is calculated according to the following formula:

$$\mathscr{I}_{\mathbb{a}}^{i}[j] = \mathscr{L}_{\mathbb{a}}[(\mathfrak{b}_i + j) mod (32 - r)], 0 \leq j \leq v_i - 1 \tag{3}$$

$\mathfrak{b}_i$ $(0 \leq i \leq u-1)$ is a parameter of READ, which is determined at the beginning of the algorithm. In order to recover the global candidate super point from REC, $\mathfrak{b}_i$ meets the following conditions when setting:

- $\mathfrak{b}_0 = 0$
- $\mathfrak{b}_i < \mathfrak{b}_{i+1} < 31 - r, i \in [0, u-2]$
- $\mathfrak{b}_{i+1} < \mathfrak{b}_i + v_i - 1, i \in [0, u-2]$
- $\mathfrak{b}_{u-1} + v_{u-1} > 31 - r$

The above conditions ensure that each bit in $\mathscr{L}_{\mathbb{a}}$ appears in at least one $\mathscr{I}_{\mathbb{a}}^{i}$, and that there are the same bits between two adjacent $\mathscr{I}_{\mathbb{a}}^{i}$ (associating with the same bit in $\mathscr{L}_{\mathbb{a}}$). When restoring global candidate super points, READ extracts the associating bits of $\mathscr{L}_{\mathbb{a}}$ from all $\mathscr{I}_{\mathbb{a}}^{i}$ to recover $\mathscr{L}_{\mathbb{a}}$, and reduces the number of global candidate super points by using the repeated bits between two adjacent $\mathscr{I}_{\mathbb{a}}^{i}$.

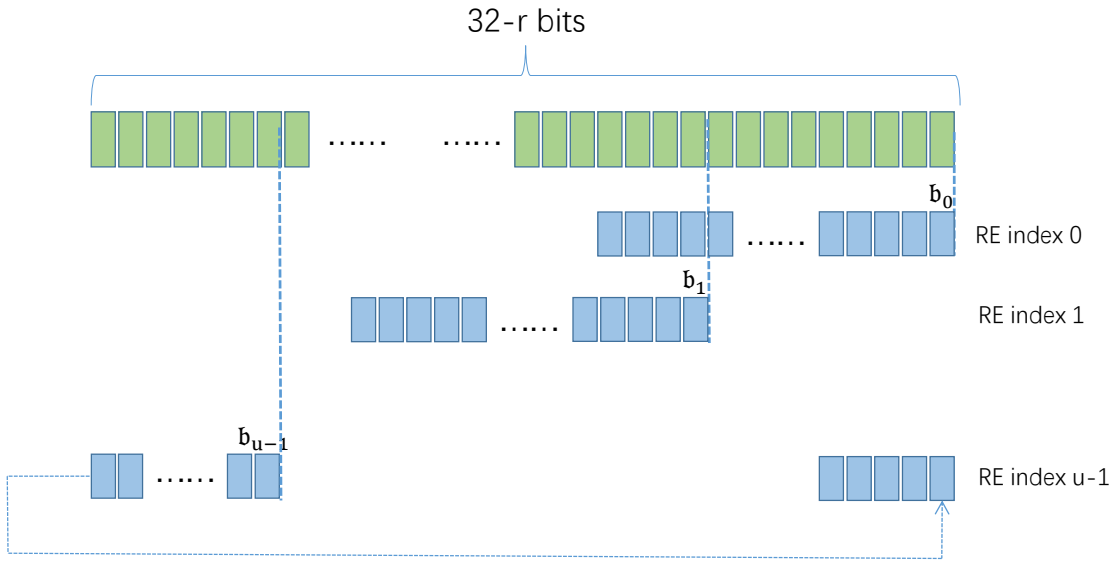

Figure 4. Locate RE by the left part of IP address.

RE estimator only determine whether the host is a global candidate super point, but cannot give an estimate of the cardinality. Therefore, READ uses LE to estimate the cardinality of each global candidate super points.

READ uses LE array of $\hat{u}$ rows and $\hat{v}$ columns to record the opposite hosts of $\mathtt{a}$, as shown in Figure 5.

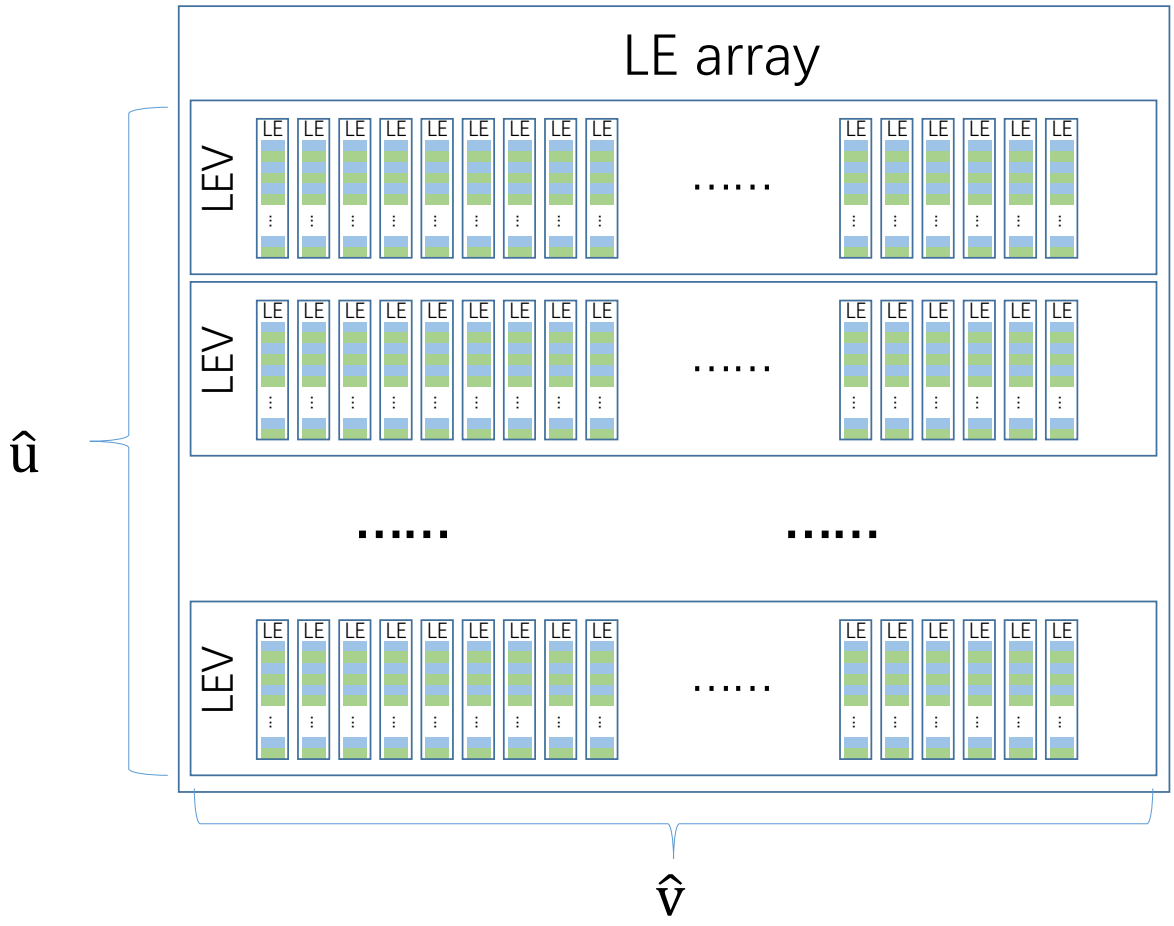

Figure 5. Structure of LE array.

LE vector (LEV) contains $\hat{u}$ LE, and LEA contains u LEV. Each observation node has a LEA, and the LEA at all observation nodes has the same structure. Let $\mathbb{L}^{\mathfrak{l}}$ denote the LEA at the $\mathfrak{l}$-th observation node, and $\mathbb{L}^{\mathfrak{l}}_{i,j}$ denote the j-th LE in the i-th LEV of $\mathbb{L}^{\mathfrak{l}}$.

For each $\mathtt{a}$ in $\mathbb{A}$, READ selects one LE from each LEV of LEA to record the opposite hosts of $\mathtt{a}$. READ maps $\mathtt{a}$ to $\hat{u}$ LE in LEV with $\hat{u}$ random hash functions. READ uses the hash function $\mathbb{h}^{LEA}_i(\mathtt{a})$ when mapping $\mathtt{a}$ to a LE in the i-th LEV, where $\mathbb{h}^{LEA}_i(\mathtt{a}) \in [0, \hat{v}-1], 0 \leq i \leq \hat{u}-1$. The observation node $\mathscr{O}_{\mathfrak{l}}$ not only updates $\mathbb{R}^{\mathfrak{l}}$, but also $\mathbb{L}^{\mathfrak{l}}$ when scanning $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}}$.

Algorithm 1 describes how READ scans IP address pairs in one observation node. READ first determines the size of REC and LEA according to the parameters, allocates the memory needed by REC and LEA, and initializes the counters of all RE and LE. Then, it starts scanning each IP address pair in $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}}$ and updates REC and LEA. When scanning IP address pairs $< \mathtt{a}, \mathbb{b} >$, READ selects a REA from the REC by using r bits on the right side of $\mathtt{a}$, and extracts $32-r$ bits on the left side of $\mathtt{a}$ as $\mathscr{L}_{\mathtt{a}}$. Then, the index of RE in each REV is determined according to $\mathscr{L}_{\mathtt{a}}$. Here, the index of RE refers to the location of RE in REV and takes the value between $[0, 2^{v_i}-1]$, where 2^{v_i} is the number of RE contained in the REV. For the i-th REV, parameter $\mathfrak{b}_i$ specifies the bits in $\mathscr{L}_{\mathtt{a}}$ associating with the first bit of the RE index. After the index value of RE is obtained, the RE is updated with $\mathbb{b}$. Compared with updating $\mathbb{R}^{\mathfrak{l}}$, updating $\mathbb{L}^{\mathfrak{l}}$ is much simpler, because $\mathbb{L}^{\mathfrak{l}}$ is only used to estimate the cardinality and does not need to restore the global candidate super point.

Algorithm 1 scanIPair.

Input: r,u, $\{v_0, v_0, \cdots, v_{u-1}\}$, $\{\mathfrak{b}_0, \mathfrak{b}_1, \cdots, \mathfrak{b}_{u-1}\}$, $\hat{u}$, $\hat{v}$, $|\mathbb{C}|$, $\{\mathbb{h}^{LEA}_0, \mathbb{h}^{LEA}_1, \cdots, \mathbb{h}^{LEA}_{\hat{u}-1}\}$, $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}}$
Output: $\mathbb{R}^{\mathfrak{l}}$, $\mathbb{L}^{\mathfrak{l}}$

```
Init R^l
Init L^l
for < a, b >∈ S^pair_{T,l} do
    k ← right r bits of a
    L_a ← left 32-r bits of a
    for i ∈ [0,u−1] do
        j=0
        for i_1 ∈ [0, v_i − 1 do
            j = j + (L_a[(b_i + i_1)mod(32−r)] << i_1)
        end for
        Update R^l_{i,j,k} with b
    end for
    for i ∈ [0, û − 1] do
        j= h^LEA_i(a)
        Update L^l_{i,j} with b
    end for
end for
return R^l, L^l
```

After the observation node scans all IP address pairs in $\mathbb{S}^{pair}_{\mathscr{T},\mathfrak{l}}$, $\mathbb{R}^{\mathfrak{l}}$ and $\mathbb{L}^{\mathfrak{l}}$ record the information of opposite hosts. By collecting $\mathbb{R}^{\mathfrak{l}}$ and $\mathbb{L}^{\mathfrak{l}}$ from all observation nodes, the global candidate super points can be recovered and the cardinalities of candidate super points can be estimated.

The next section describes how READ recovers global candidate super points in a distributed environment.

4.3. Generate Candidate Super Points

The master data structure at the observation node consists of two parts: REC and LEA. REC is used to recover global candidate super points, which has the advantage of

less memory consumption; LEA is used to estimate cardinality, which has the advantage of high estimation accuracy. Each observation node can only observe part of the opposite hosts. In order to detect the super points accurately, it is necessary to collect the opposite hosts information recorded by each observation node on the global server. In this paper, the super points detected from IP address pairs of all observation nodes are called as global super points, and the generated global candidate super points are called global candidate super points. When generating global candidate super points, only RECs are collected from each observation node, as shown in Figure 6.

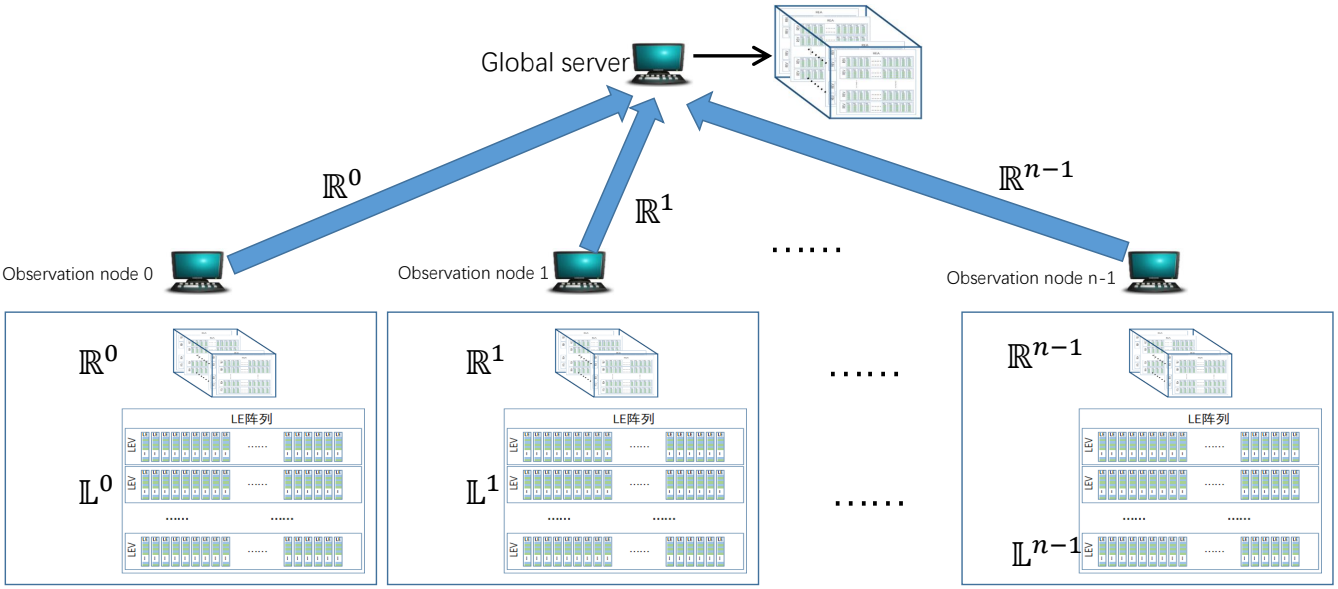

Figure 6. Collect REC from observation nodes.

After each observation node has scanned all IP address pairs in a time window, only the REC needs to be sent to the global server. The global server merges all the collected REC. The merging method is to merge the RE of different observation nodes in a "bit or" manner. In this paper, the way of combining according to "bit or" is called external merging, and the way of combining according to "bit and" is called internal merging. External merger of RE is defined as follows:

Definition 3 (RE Out merging)**.** *All bits of two RE generate a new RE according to the operation of "bit or".*

In this paper, when the operand of the operator "$\oplus$" is two RE or two LE, it means to out merge the two RE or LE; when the operand of the operator "$\odot$" is two RE or two LE, it means to inner merge the two RE or LE.

The REC of all observation nodes are merged on the global server by outer merging, which ensures that any bit in the REC is still 1 in the merged global REC as long as it is set to 1 at any one observation node. Since RE uses bits to record the occurrence of opposite host, the global REC generated by outer merging contains the opposite information recorded by all observation nodes.

In this paper, the REC used to restore the global candidate super points on the global server is called as the global REC. The global REC has the same structure as the REC at all observation nodes. The global REC and the REC of all observation nodes are merged according to outer merging. There are two methods to get the global REC:

1. Before merging the REC, the global server initializes a REC with the same structure as the REC at the observation nodes, and sets all bits in the initialized REC to 0. Then,

the REC on the global server is merged with the REC on all observation nodes one by one, and the results are saved to the global REC.

2. The global server takes the REC from the first observation node as the global REC, then merges the global REC with the REC from the remaining observation nodes, and saves the results to the global REC.

Among the two methods for merging global REC, method 2 is less computational than method 1, because method 2 does not need to re-initialize REC. In this paper, method 2 is used to merge the REC of observation nodes into the global REC. Let $\mathbb{R}$ denote the global REC, and $\mathbb{R}_{i,j,k}$ denote the j-th RE of the i-th REV in the k-th REA of $\mathbb{R}$. Assuming that the REC on $\mathscr{O}_\theta$ is first received as one on the global server, Algorithm 2 describes the REC merging process on the global server.

Algorithm 2 Out Merging REC.

Input: $\mathfrak{n}$, $\{\mathbb{R}^0, \mathbb{R}^1, \cdots, \mathbb{R}^{n-1}\}$, r, u, $\{v_0, v_1, \cdots, v_{u-1}\}$
Output: $\mathbb{R}$

```
ℝ ← ℝ^0
for 𝔩 ∈ [1, 𝔫 − 1] do
    for k ∈ [0, 2^r − 1] do
        for i ∈ [0, 2^u − 1] do
            for j ∈ [0, 2^{v_i} − 1] do
                ℝ_{i,j,k} ← (ℝ_{i,j,k} ⊕ ℝ^𝔩_{i,j,k})
            end for
        end for
    end for
end for
Return ℝ
```

The first line of Algorithm 2 takes the received $\mathbb{R}^0$ as the global REC after the first merge, and then merges the remaining $\mathfrak{n}-1$ observation nodes into the global REC. After merging the REC at all observation nodes, algorithm 2 outputs the global REC.

READ recovers the global candidate super points from each REA of the global REC in turn. For the k-th REA of the global REC (denoted as $\mathscr{A}_k$), READ calculates the global candidate super points in it by the following two steps:

1. Find out all RE in $\mathscr{A}_k$ whose estimating cardinality is greater than the threshold.
2. From the candidate RE, 32-r bits on the left of the candidate super point are recovered, and then concatenate with the right r bits represented by k to get the complete global candidate super point.

The above Step 1 only needs to scan all RE in $\mathscr{A}_k$ once to get a candidate RE. Let $\mathfrak{C}i = \{\mathfrak{c}_0^i, \mathfrak{c}_1^i, \mathfrak{c}_2^i, \cdots\}$ represent the index of the candidate RE in the i-th REV of $\mathscr{A}_k$. Equation (3) shows that the index of the candidate RE in $\mathfrak{C}i$ comes from the bits of certain IP address. At the same time, as can be seen from Figure 4, if the two indexes $\mathfrak{c}_x^i$ and $\mathfrak{c}_y^{((i+1)mod(u))}$ of two adjacent row, i and $(i+1)mod(u)$ are from the same IP address, then they have $\mathfrak{b}_i + v_i - \mathfrak{b}_{((i+1)mod(u))}$ bits are the same. Conversely, if the left $\mathfrak{b}_i + v_i - \mathfrak{b}_{((i+1)mod(u))}$ bits of $\mathfrak{c}_x^i$ are different from the right $\mathfrak{b}_i + v_i - \mathfrak{b}_{(}(i+1)mod(u))$ bits of $\mathfrak{c}_y^{((i+1)mod(u))}$, then $\mathfrak{c}_x^i$ and $\mathfrak{c}_y^{((i+1)mod(u))}$ certainly do not come from the same IP address. When the u RE indexes comes from the same IP address, the u RE indexes are called a candidate RE tuple. Inner merge these u RE in a candidate RE tuple. If the estimated value of the inner merged RE still exceeds the threshold, the candidate RE tuple come from a global candidate hyper point.

When the candidate RE tuple comes from a global candidate super point, the candidate RE tuple can recover 32-r bits to the left part of the global super point. From the setting requirement of parameter $\mathfrak{b}_i$, if the RE indexes in a candidate RE tuple comes from the same IP address $\mathfrak{a}$, any bit of $\mathscr{L}_\mathfrak{a}$ will appear at least once in the u different candidate RE

indexes. Therefore, 32-r bits of $\mathscr{L}_{\mathfrak{a}}$ can be recovered from the candidate RE tuple. Then, a global candidate super point is obtained by concatenation with k, i.e., $(\mathscr{L}_{\mathfrak{a}} << r) + k$.

Depth traversal can be used to calculate all candidate RE tuples from $\mathfrak{C}i$. For example, suppose that the parameters of REC are set to r = 2, u = 3, $v_0 = v_1 = v_2 = 14$, $\mathfrak{b}_0$=0, $\mathfrak{b}_1$=10, $\mathfrak{b}_2$=20, the candidate RE indexes of $\mathscr{A}_2$ is $\mathfrak{C}0 = \{\mathfrak{c}_0^0, \mathfrak{c}_1^0, \mathfrak{c}_2^0\}$, $\mathfrak{C}1 = \{\mathfrak{c}_0^1, \mathfrak{c}_1^1\}$, $\mathfrak{C}2 = \{\mathfrak{c}_0^2, \mathfrak{c}_1^2, \mathfrak{c}_2^2\}$. The number values of some candidate RE are as follows:

- $\mathfrak{c}_0^0 =$ 1100 010101 0101
- $\mathfrak{c}_0^1 =$ 1100 011001 0101 ,
  $\mathfrak{c}_1^1 =$ 1110 010001 1100
- $\mathfrak{c}_0^2 =$ 1001 011101 1110 ,
  $\mathfrak{c}_1^2 =$ 0101 000101 1110 .

In the above example, $\mathfrak{b}_i$+v_i-$\mathfrak{b}_{i+1}$=4, that is, the candidate RE indexes in the two adjacent $\mathfrak{C}i$ determines whether it comes from the same IP address by the four bits on the left and the four bits on the right (the gray part in the RE index). When the candidate RE tuple is calculated by depth-first method, the candidate RE tuple is empty at the beginning, and then the first RE number is $\mathfrak{c}_0^0$. Test whether $\mathfrak{c}_0^0$ and $\mathfrak{c}_0^1$ come from the same IP address, as shown in Figure 7.

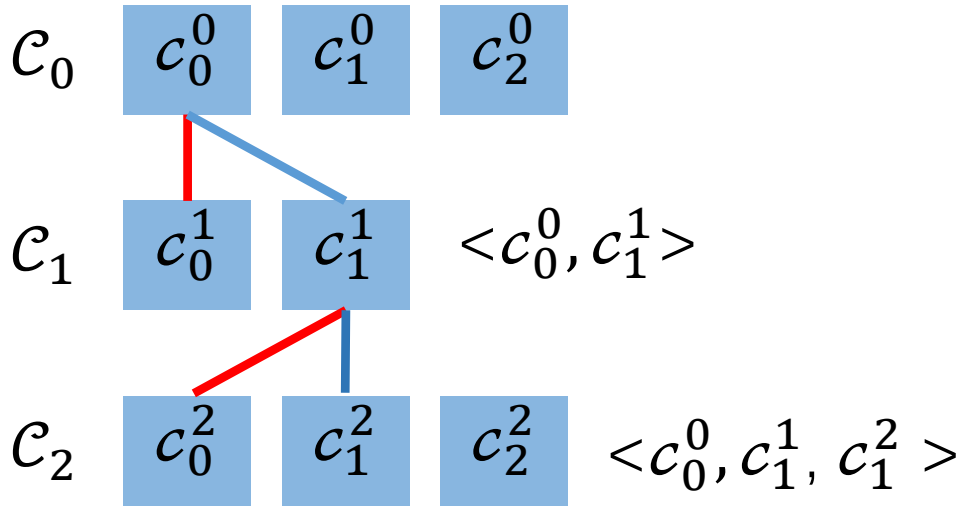

Figure 7. Example of restoring LP with depth-first method.

The four bits on the left of $\mathfrak{c}_0^0$ are different from the four bits on the right of $\mathfrak{c}_0^1$, so $\mathfrak{c}_0^0$ and $\mathfrak{c}_0^1$ come from different IP addresses. Then, test $\mathfrak{c}_0^0$ and $\mathfrak{c}_1^1$. The four bits on the left side of $\mathfrak{c}_0^0$ are the same as the four bits on the right side of $\mathfrak{c}_1^1$, so $\mathfrak{c}_1^1$ is added to the candidate RE tuple. Then, find the RE index from $\mathfrak{C}2$, which comes from the same IP address with $\mathfrak{c}_1^1$. In $\mathfrak{C}2$, the four bits on the right side of $\mathfrak{c}_0^2$ are the same as the four bits on the left side of $\mathfrak{c}_1^1$, but the four bits on the left side of $\mathfrak{c}_0^2$ are not equal to the four bits on the right side of $\mathfrak{c}_0^0$, so $\mathfrak{c}_0^2$ cannot form a candidate RE tuple with $\mathfrak{c}_0^0$ and $\mathfrak{c}_1^1$. In $\mathfrak{C}2$, not only are the four bits on the right side the same as the four bits on the left side of $\mathfrak{c}_1^1$, but also the four bits on the left side of $\mathfrak{c}_1^2$ the same as the four bits on the right side of $\mathfrak{c}_0^0$. Therefore, $< \mathfrak{c}_0^0, \mathfrak{c}_1^1, \mathfrak{c}_1^2 >$ constitutes a candidate RE tuple.

From the values of $\mathfrak{c}_0^0$, $\mathfrak{c}_1^1$ and $\mathfrak{c}_1^2$, it can be seen that the RE associating with the candidate RE tuple is $\mathbb{R}^{\mathfrak{l}}_{0,12629,2}$, $\mathbb{R}^{\mathfrak{l}}_{1,14620,2}$, $\mathbb{R}^{\mathfrak{l}}_{2,5214,2}$. If the cardinality estimated from the inner merge RE, $\mathbb{R}^{\mathfrak{l}}_{0,12629,2} \odot \mathbb{R}^{\mathfrak{l}}_{1,14620,2} \odot \mathbb{R}^{\mathfrak{l}}_{2,5214,2}$, still over the threshold, 30 bits of the left part of $\mathfrak{a}$ can be recovered from $< \mathfrak{c}_0^0, \mathfrak{c}_1^1, \mathfrak{c}_1^2 >$: "000101 1110 010001 1100 010101 0101 ". $\mathscr{A}_2$ is the 2-th REA in REC. The associating binary format is "10". Thus, the global candidate super point is "000101 1110 010001 1100 010101 0101 10".

All REA in global REC are processed in the above way. Because the number of RE counters is small (for IPv4 address, there are only eight counters), so it is faster to scan

REA and calculate the candidate RE number. Furthermore, each RE only takes up one byte of space, so REC takes up less memory and reduces the amount of data transmitted between observation nodes and the global server. However, the cardinalities of the global candidate super points cannot be estimated by RE. Estimating the cardinality requires the use of the opposite host information stored in LEA. The next section describes how to collect the opposite host information stored in LEA from the observation nodes, estimate the cardinalities of the global candidate super points, and filter out the super points.

4.4. Estimate Cardinalities of Candidate Super Points

The LEA at each observation node is used for estimating the cardinality of global candidate super points. A simple way is to send all LEAs at each observation node to the global server, and then merge all LEA of observation nodes on the global server in a "bit or" manner to get the global LEA.

In this paper, when the operand of "$\sum$" is the LE or RE set, it means that all LE or RE in the set are merged by outer merging method; when the operand of "$\prod$" is the LE or RE set, it means that all LE or RE in the set are merged by inner merging method.

Merging LEA of all observation nodes on the global server in the way of outer merging is equivalent to sending IP address pairs directly to the global server to update the global LEA. Because LE outer merging guarantees that any bit in the global LEA will remain 1 as long as it is set to 1 at one or more observation nodes.

After the global LEA is generated, the cardinalities of global candidate super points can be estimated according to the global LEA. Let $\mathfrak{q}$ denote a global candidate super point, $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$ denote the LE of $\mathfrak{q}$ in the i-th LEV of the $\mathfrak{l}$-th observation node, i.e., $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q}) = \mathbb{L}_{i,j}^{\mathfrak{l}}$, $j = \mathbb{h}_i^{LEA}(\mathfrak{q})$. Using hash functions $\mathbb{h}_i^{LEA}(\mathfrak{q})$, it is easy to find these LEs used by $\mathfrak{q}$ from the global LEA.

Let $\mathfrak{B}_i(\mathfrak{q})$ denote the LE associating with $\mathfrak{q}$ in the first LEV of the global LEA. Since global LEA is obtained by combining LEA from all observation nodes, $\mathfrak{B}_i(\mathfrak{q}) = \sum_{l=0}^{n-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$. The $\hat{u}$ LE of $\mathfrak{q}$ on the global LEA are merged into $\overline{\overline{\mathfrak{B}(\mathfrak{q})}} = \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i(\mathfrak{q})$. Let $|\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$ denote the number of bits with value "1" in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$. The cardinality of $\mathfrak{q}$ is estimated based on $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$ by Equation (1). If the estimated result is larger than the threshold, $\mathfrak{q}$ is reported as a super point.

Although the above method avoids sending all IP addresses to the global server, it still needs to send the complete LEA to the global server. In order to improve the accuracy of cardinality estimating, the parameters of LEA are set to larger values. For example, when $\hat{u} = 5$, $\hat{v} = 2^{15}$, $|\mathbb{C}| = 2^{14}$, LEA is 320 MB in size. When estimating cardinalities, each observation node needs to send 320MB of data to the global server.

When estimating the cardinality of global candidate super point $\mathfrak{q}$, only $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$ is needed. Based on this principle, READ first sends the global candidate super points to each observation node from the global server, and then each observation node sends these LE relating with candidate super points back to the global server, as shown in Figure 8.

In Figure 8, $\mathbb{Q} = \{\mathfrak{q}_0, \mathfrak{q}_1, \mathfrak{q}_2, \cdots, \mathfrak{q}_{\mathfrak{w}-1}\}$ denotes the set of global candidate super points, $\overline{\mathfrak{B}^{\mathfrak{l}}}$ denotes the set of LE used to estimate cardinalities of global candidate super points in $\mathbb{Q}$ on the observation node $\mathfrak{l}$. For global candidate super point $\mathfrak{q}$, there are $\hat{u}$ LE associating with it, i.e., $\{\mathfrak{B}_0^{\mathfrak{l}}(\mathfrak{q}), \mathfrak{B}_1^{\mathfrak{l}}(\mathfrak{q}), \cdots, \mathfrak{B}_{\hat{u}-1}^{\mathfrak{l}}(\mathfrak{q})\}$. READ does not send all of the $\hat{u}$ LE to the global server, but the result of internal merging , $\overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})} = \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$.In Figure 8, $\overline{\mathfrak{B}^{\mathfrak{l}}} = \{\overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q}_0)}, \overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q}_1)}, \overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q}_2)}, \cdots, \overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q}_{\mathfrak{w}-1})}\}$ is the LE set to be sent to the global server on the $\mathfrak{l}$-th observation node.

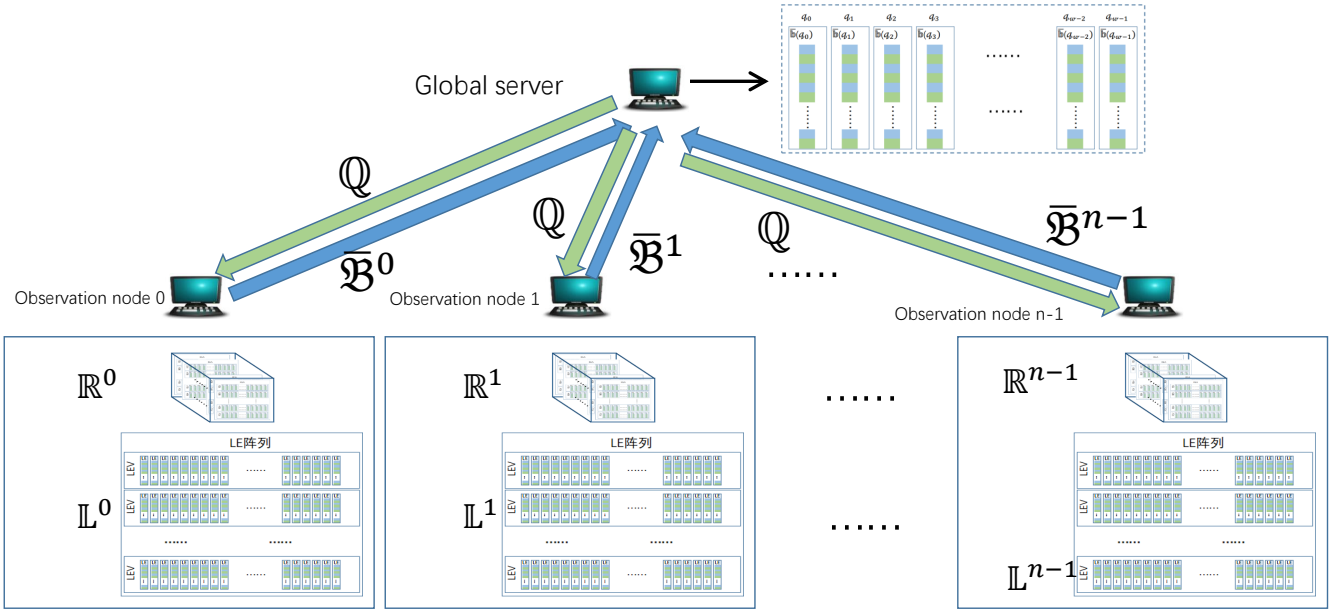

Figure 8. Collect candidate LE in a distributed environment.

On the global server, $\overline{\mathfrak{B}(\mathfrak{q})} = \sum_{\mathfrak{l}=0}^{\mathfrak{n}} \overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})}$, which is used for estimating the cardinality of $\mathfrak{q}$, is obtained by outer merging all $\overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})}$. Let $|\overline{\mathfrak{B}(\mathfrak{q})}|$ denote the number of bits with value "1" in $\overline{\mathfrak{B}(\mathfrak{q})}$. Theorem 1 shows that $\overline{\mathfrak{B}(\mathfrak{q})}$ can more accurately estimate the cardinality of $\mathfrak{q}$ than $\overline{\mathfrak{B}(\mathfrak{q})}$.

Theorem 1. *For global candidate super point* $\mathfrak{q}$*, let* $\mathbb{S}^{\mathbb{B}}_{\mathscr{T},\mathfrak{q}}$ *denote the set of opposite hosts of* $\mathfrak{q}$ *passing through all observation nodes in time window* $\mathscr{T}$*,* $\mathfrak{B}(\mathfrak{q})$ *denote a LE after scanning* $\mathbb{S}^{\mathbb{B}}_{\mathscr{T},\mathfrak{q}}$*, and* $|\mathfrak{B}(\mathfrak{q})|$ *denote the number of bits with value "1" in* $\mathfrak{B}(\mathfrak{q})$*. Then, these bits with value "1" in* $\mathfrak{B}(\mathfrak{q})$ *are still with value "1" in* $\overline{\mathfrak{B}(\mathfrak{q})}$ *and* $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$*. Furthermore,* $|\mathfrak{B}(\mathfrak{q})| \leq |\overline{\mathfrak{B}(\mathfrak{q})}| \leq |\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$.

Proof of Theorem 1. When a bit in $\mathfrak{B}(\mathfrak{q})$ has value "1", there exists an IP address pair $< \mathfrak{q}, \mathbb{b} >$ in $\mathbb{S}^{\mathbb{B}}_{\mathscr{T},\mathfrak{q}}$ to set the bit to "1". In global LEA, $\mathbb{b}$ sets all the bits of $\hat{u}$ LE associating with $\mathfrak{q}$. After inner merging in LE, the bit is "1" in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$. At the same time, $\mathbb{b}$ will appear on at least one observation node and set all the bits of $\hat{u}$ LE associating with $\mathfrak{q}$ to "1". Since the bit is "1" in at least one $\overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})}$, the bit is still "1" after outer merging on the global server. So, $|\mathfrak{B}(\mathfrak{q})| \leq |\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$ and $|\mathfrak{B}(\mathfrak{q})| \leq |\overline{\mathfrak{B}(\mathfrak{q})}|$. The next step is to proof that $|\overline{\mathfrak{B}(\mathfrak{q})}| \leq |\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$.

Let $\mathfrak{B}_i(\mathfrak{q}) = \sum_{\mathfrak{l}=0}^{n-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$, then $\overline{\overline{\mathfrak{B}(\mathfrak{q})}} = \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i(\mathfrak{q}) = \prod_{i=0}^{\hat{u}-1} \sum_{l=0}^{n-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$. Let $\overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})} = \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$, then $\overline{\mathfrak{B}(\mathfrak{q})} = \sum_{\mathfrak{l}=0}^{n-1} \overline{\mathfrak{B}^{\mathfrak{l}}(\mathfrak{q})} = \sum_{\mathfrak{l}=0}^{n-1} \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$. To proof that $|\overline{\mathfrak{B}(\mathfrak{q})}| \leq |\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$ is equivalent to proof that the number of bits with value "1" in $\sum_{\mathfrak{l}=0}^{n-1} \prod_{i=0}^{\hat{u}-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$ is no more than the number of bits with value "1" in $\prod_{i=0}^{\hat{u}-1} \sum_{\mathfrak{l}=0}^{n-1} \mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$. $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$ is a LE and the number of bits in all $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$ are the same. Let $\beta_i^{\mathfrak{l}}$ denote an arbitrary bit in $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$. All $\beta_i^{\mathfrak{l}}$ in different observation nodes could be written as an array in the following format:

$$\beta = \begin{bmatrix} \beta_0^0 & \cdots & \beta_0^{\mathfrak{n}-1} \\ \vdots & \ddots & \vdots \\ \beta_{\hat{u}-1}^0 & \cdots & \beta_{\hat{u}-1}^{\mathfrak{n}-1} \end{bmatrix}$$

In β, $\prod_{i=0}^{\hat{u}-1} \sum_{\mathfrak{l}=0}^{n-1} \beta_i^{\mathfrak{l}}$ represents that "bit or" operations are performed on each line, and then "bit and" operations are performed on the results; $\sum_{\mathfrak{l}=0}^{n-1} \prod_{i=0}^{\hat{u}-1} \beta_i^{\mathfrak{l}}$ represents that "bit and" operations are performed on each line, and then "bit or" operations are performed on the results.

When $\prod_{i=0}^{\hat{u}-1}\sum_{\mathfrak{l}=0}^{n-1}\beta_i^{\mathfrak{l}}=0$, at least one row has all bits equal to "0", and the result of "bit and" operation for each column is also 0, then $\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}=\sum_{\mathfrak{l}=0}^{n-1}0=0$. When $\prod_{i=0}^{\hat{u}-1}\sum_{\mathfrak{l}=0}^{n-1}\beta_i^{\mathfrak{l}}=1$, there is no row whose bits are all "0". However, $\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}$ may still be 0. Because when each column of β contains at least one bit with value "0", then $\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}=\sum_{\mathfrak{l}=0}^{n-1}0=0$. At this time, each row may contains one or more bits with value "1". For example, when n=3,$\hat{u}=3$,$\beta=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$, $\prod_{i=0}^{\hat{u}-1}\sum_{\mathfrak{l}=0}^{n-1}\beta_i^{\mathfrak{l}}=1$, but$\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}=0$.

When $\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}=1$, $\prod_{i=0}^{\hat{u}-1}\sum_{\mathfrak{l}=0}^{n-1}\beta_i^{\mathfrak{l}}$ also equals to 1. As when $\sum_{\mathfrak{l}=0}^{n-1}\prod_{i=0}^{\hat{u}-1}\beta_i^{\mathfrak{l}}=1$, at least one column in β has all bits with value "1". Then, there is no row in β whose bits are all "0". As $\beta_i^{\mathfrak{l}}$ is an arbitrary bit in $\mathfrak{B}_i^{\mathfrak{l}}(\mathfrak{q})$, then:

- When a bit has value "1" in $\overline{\mathfrak{B}(\mathfrak{q})}$, the bit has value "1" in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$;
- When a bit has value "0" in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$, the bit has value "0" in $\overline{\mathfrak{B}(\mathfrak{q})}$;
- When a bit has value "1" in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$, the bit may has value "0" in $\overline{\mathfrak{B}(\mathfrak{q})}$

So the number of bits with value "1" in $\overline{\mathfrak{B}(\mathfrak{q})}$ is no more than that in $\overline{\overline{\mathfrak{B}(\mathfrak{q})}}$ and $|\mathfrak{B}(\mathfrak{q})|\leq|\overline{\mathfrak{B}(\mathfrak{q})}|\leq|\overline{\overline{\mathfrak{B}(\mathfrak{q})}}|$. □

LE estimates cardinality based on the number of bits with value "1". Theorem 1 shows that the number of bits with value "1" in $\overline{\mathfrak{B}(\mathfrak{q})}$ is closer to the number of bits with value "1" in the LE which is used by $\mathfrak{q}$ exclusively. So, the accuracy of estimating cardinality by $\overline{\mathfrak{B}(\mathfrak{q})}$ is better.

READ not only does not need to transfer the entire LEA to the global server, but also has a higher accuracy in estimating cardinalities of global candidate super points. When estimating cardinalities, the amount of data transmitted between each observation node and the global server is $(32*\mathfrak{w}+|\mathbb{C}|*\mathfrak{w})$ bits, where $\mathfrak{w}$ is the number of candidate super points recovered by REC. $32*\mathfrak{w}$ is the data size of global candidate super points transmitting to each observation node from the global server, and $|\mathbb{C}|*\mathfrak{w}$ is the data size of LE of candidate super points that transmitting to the global server from each observation node. When $(32*\mathfrak{w}+|\mathbb{C}|*\mathfrak{w})<\hat{u}*\hat{v}*|\mathbb{C}|$, the data transmission between an observation node and the global server is less than the data transmission of the entire LEA. Global candidate super points account for only a small portion of all IP addresses, usually hundreds to thousands. In order to improve the estimation accuracy, the value of $\hat{u}*\hat{v}$ will be more than tens of thousands. So, READ reduces the amount of data transmitted between observation nodes and the global server. READ can also apply more powerful counters to replace bits in RE and LE to realize the detection of super points under a sliding time window as discussed in the next section.

5. Distributed Super Points Detection under Sliding Time Window

READ only scans IP address pairs at each observation node, so only a sliding window counter is needed to record opposite hosts incrementally at the observation node. The master data structure at the observation node consists of two parts: REC and LEA. The estimators of REC and LEA are RE and LE, while the counters used by RE and LE are bits. So, the master data structure at the observation node can be regarded as a set of bits. Using counter DR[20] or AT[27] under sliding window instead of bit in REC and LEA at each observation node, distributed super points detection under sliding window can be realized.

The counter under the sliding window needs to be updated. After all LE associating with the global candidate super points are sent to the global server, the observation node can start to update the sliding counter. At the end of each time window, the REC on the global server is generated by these REC collecting from all observation nodes, there is no need to update it.

Under the sliding time window, the observation node only needs to send the active state of the counter to the global server, that is, at the end of the time window, each sliding window counter can be changed into a bit: 0 for inactivity, 1 for activity. Therefore, under sliding time window, the traffic between observation nodes and the global server is the same as that under discrete time window.

READ can be quickly deployed to distributed networks. For example, suppose that network $\mathbb{A}$ and network $\mathbb{B}$ communicate through three different routers. An IP address pair in the form of < $\mathtt{a}$, $\mathtt{b}$> can be extracted from the IP packet on each router. On the observation node of each router, select REs from RE cube and LEs from LE array according to $\mathtt{a}$; update the selected REs and LEs according to $\mathtt{b}$. At the end of the time window, send the RE cubes on the three router observation nodes to the global server for merging, and generate candidate super points from the merged RE cubes. Then, the candidate super points are sent to these three router observation nodes for LEs selection. Finally, the global server collects the LEs of candidate super points from three router observation nodes and filters out the super points. The following section will evaluate READ with high-speed network traffic.

6. Experiments and Analysis

In order to test the performance of READ, four groups of high-speed network traffic are used to carry out experiments in this section. The experiment analyzes READ from the aspects of detection error rate, memory usage and running time. The experiment compared READ with DCDS, VBFA, CSE and SRLA.

6.1. Experiment Data

In this paper, four groups of high-speed network traffic are used. Two of the four sets of data come from the 10 Gb/s Caida[28]. The other two groups are from the network boundary of the 40Gb/s CERNET in Nanjing network[29].

The Caida data acquisition dates are February 19, 2015 and January 21, 2016 (denoted by *Caida* 2015_2_19 and *Caida* 2016_01_21), and the data acquisition dates of the two groups of CERNET Nanjing network were October 23, 2017 and March 8, 2018 (denoted by *IPtas* 2017_10_23 and *IPtas* 2018_03_08). The collection time of the four groups of data is one hour from 13:00. The collected data are raw IP Trace. Caida data collected Trace between Seattle and Chicago. In this paper, the IP on Seattle side is defined as $\mathtt{a}$, and the IP on Chicago side is defined as $\mathtt{b}$. IPtas data collects traces between CERNET Nanjing network and other networks. In this paper, the IP in Nanjing network is $\mathtt{a}$, and in the other network is $\mathtt{b}$.

In the experiment of this section, the length of time window is 5 min, and the threshold of super point is set to 1024. Therefore, each group of experimental data contains 12 time windows. Table 2 lists the statistical information of each experimental data. The number of $\mathtt{a}$ in Caida data is more than the number of $\mathtt{a}$ in IPtas data, which is 1.85 times more on average. However, the average cardinality per $\mathtt{a}$ in Caida data is less than that in IPtas data, only 21.389% of the latter. The number of packets per second determines the number of IP address pairs that need to be processed per second. Therefore, packet speed (in millions of packets per second, Mpps) is a key attribute. As can be seen from Table 2, the average packet speed of IPtas data is 3.89 times that of Caida data. Therefore, Caida data and IPtas data represent two different types of network data sets, which can test the effect of the algorithm more comprehensively.

Table 2. Statistics of experiment data.

Traffic Name	Statistic Type	Number of $\mathbb{a}$	Number of $\mathbb{b}$	Number of IP Pair	Average Cardinality	Number of Packet(Mpkt)	Packet Speed(Mpps)	Number of Super Points
Caida 2015_02_19	Average	2,500,423	1,536,625	6,608,075	2.6713	268.9149	0.8964	162.1667
	Max	2,844,368	1,639,128	6,965,239	3.0884	276.8782	0.9229	178
	Min	2,026,263	1,490,879	6,241,517	2.4414	258.2578	0.8609	153
	StandardDeviation	313,920	39,868	269,719	0.252	5.8792	0.0196	7.4203
Caida 2016_01_21	Average	2,437,770	746,177	4,800,712	1.9691	322.4348	1.0748	41.9167
	Max	2,488,042	811,230	4,944,912	2.013	344.9535	1.1498	49
	Min	2,382,249	702,651	4,637,869	1.9142	303.239	1.0108	36
	StandardDeviation	34,286	32,638	118,781	0.0286	14.7145	0.049	3.1176
IPtas 2017_10_23	Average	1,262,184	1,588,792	15,163,646	12.0132	1354.1672	4.5139	598.8333
	Max	1,262,810	1,721,288	32,847,335	26.0139	1463.4874	4.8783	662
	Min	1,261,625	1,515,963	12,573,274	9.9649	1265.9158	4.2197	581
	StandardDeviation	371	49,878	5,596,915	4.431	63.054	0.2102	22.1722
IPtas 2018_03_08	Average	1,406,287	1,815,909	13,429,067	9.5422	946.4292	3.1548	527.4167
	Max	1,436,128	1,865,955	30,234,164	21.3223	1253.2099	4.1774	569
	Min	1,378,231	1,758,650	11,299,384	7.9936	890.201	2.9673	505
	StandardDeviation	18,387	30,026	5,300,542	3.7187	97.9128	0.3264	17.7787

6.2. The Purpose and Scheme of the Experiment

The experimental purposes of this paper are as follows:

- Analyze the accuracy of READ and test whether REC can accurately generate candidate super points.
- Analyze the memory occupancy and running time of READ;
- Test the number of candidate super points generated by READ and the amount of data that needs to be transmitted between each observation node and the global server.

In order to process high-speed network data in real time, this paper deploys READ, DCDS, VBFA, CSE and SRLA algorithm on GPU platform. All the experiments in this paper run on a server with GPU. The running environment is: Intel Xeon E5-2643 CPU, 125 GB memory, Nvidia Titan XP GPU, 12 GB memory, Debian Linux 9.6 operating system.

In the experiment, the parameters of REC are $r = 6$, $u = 3$, $v_0 = v_1 = v_2 = 14$; the parameters of LEA are $\hat{u} = 5$,$\hat{v} = 2^{15}$ and $|\mathbb{C}| = 2^{15}$. From the above parameters, it can be seen that REC occupies 3 MB of memory and LEA occupies 320 MB of memory. Because there is no distributed experimental data, the experiment in this section is carried out under a single node. However, from the previous analysis of READ, it can be seen that the error rate of READ in a distributed environment will not be higher than that in a single node environment.

6.3. Memory and False Rate

In order to analyze the memory and false rate of READ, this section compares READ with DCDS, VBFA, CSE and SRLA algorithm. Table 3 shows the average memory occupancy and error rate of READ and comparison algorithms in different experimental data sets. False positive rate (FPR), false negative rate (FNR) and false total rate (FTR) are three kinds of false rates. Let N represent the number of super points, N^- represent the number of super points that are not detected out by an algorithm and N^+ represent the number of hosts whose cardinalities are less than the threshold, but detected as super points by an algorithm. Then, $FPR = 100 * N^+/N\%$, $FNR = 100 * N^-/N\%$, $FTR = FPN + FNR$.

Table 3. Memory and false rate.

Experiment traffic	Algorithm name	Memory(MB)	FPR(%)	FNR(%)	FTR(%)
Caida 2015_02_19	DCDS	384.00	0.72	0.32	1.04
	VBFA	320.00	0.92	0.15	1.07
	CSE	512.00	2.02	1.26	3.28
	SRLA	320.63	0.76	0.83	1.59
	READ	323.00	0.87	0.71	1.58
Caida 2016_01_21	DCDS	384.00	0.77	0.84	1.61
	VBFA	320.00	1.78	0.40	2.18
	CSE	512.00	3.86	3.21	7.07
	SRLA	320.63	0.82	1.01	1.84
	READ	323.00	1.03	0.40	1.42
IPtas 2017_10_23	DCDS	384.00	5.00	0.00	5.00
	VBFA	320.00	5.43	0.00	5.43
	CSE	512.00	1.39	1.27	2.66
	SRLA	320.63	2.42	0.55	2.97
	READ	323.00	2.45	0.44	2.89
IPtas 2018_03_08	DCDS	384.00	5.59	0.02	5.61
	VBFA	320.00	6.56	0.00	6.56
	CSE	512.00	1.44	1.40	2.84
	SRLA	320.63	3.36	0.56	3.91
	READ	323.00	2.96	0.32	3.28

Table 3 shows that READ occupies less memory than DCDS and CSE, and only 3 MB more memory than VBFA. In terms of error rate, the error rate of READ is close to that of SRLA algorithm.

6.4. Running Time Analysis

Figure 9 shows the time of IP address pairs scanning (GScanT). The graph shows that the GScanT of READ is slightly higher than that of SRLA algorithm. However, the GScanT of each algorithm is not more than 4 s, which can process 40 Gb/s of high-speed network traffic in real time.

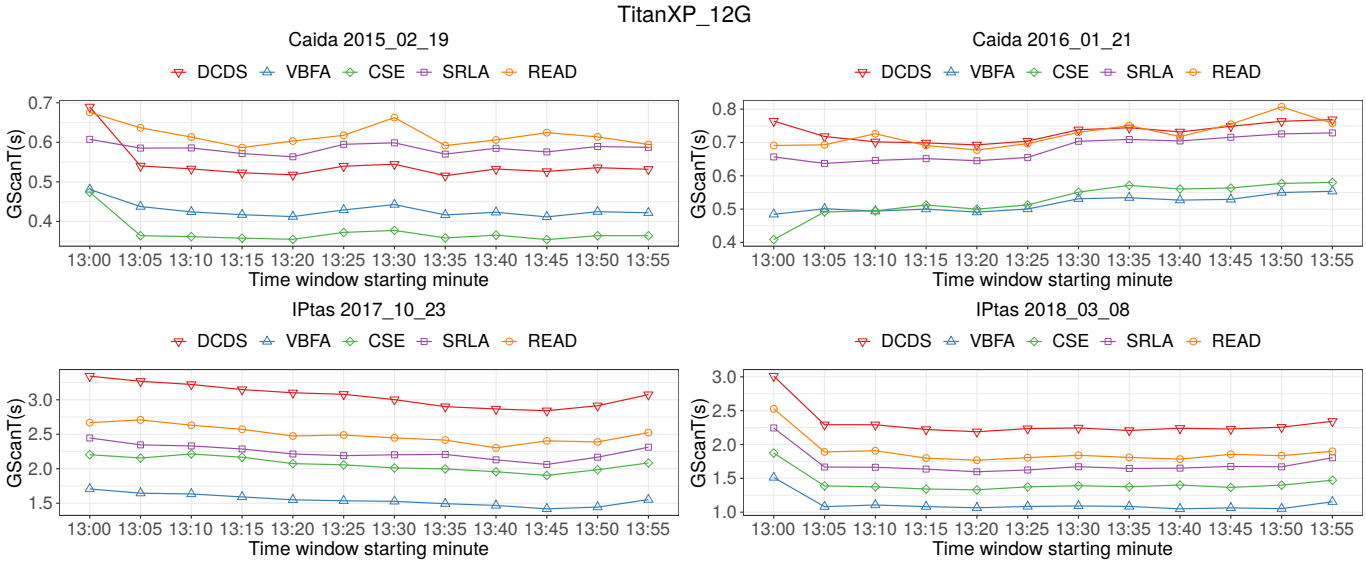

Figure 9. Time of scan IP address pair.

Figure 10 shows the time of candidate super point cardinality estimation (GEstT). The graph shows that GEstT of READ is close to DCDS, VBFA and SRLA algorithm, much lower than CSE, and GEstT of READ is not higher than 2.5 s. Therefore, READ can detect super points in real-time from 40Gb/s high-speed network.

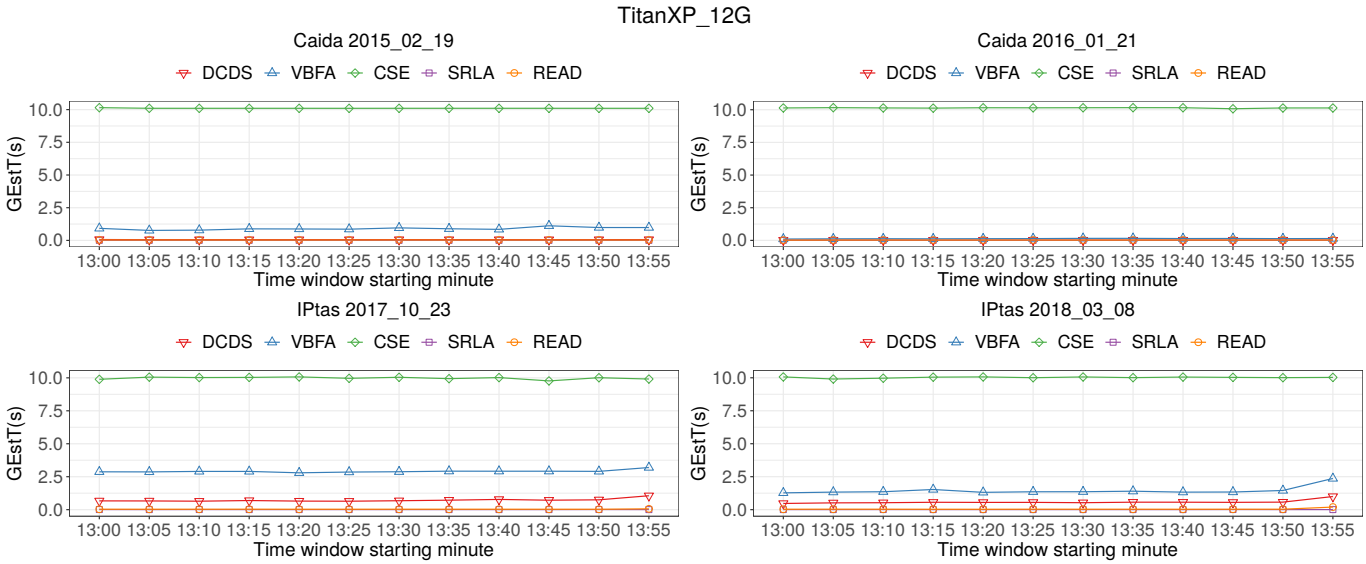

Figure 10. Time of estimate candidate super points.

6.5. Data Transmission under Distributed Environment

READ is a distributed algorithm. In a distributed environment, data will be transmitted between each observation node and the global server, including:

- REC from observation node to the global server;
- Candidate super points from the global server to each observation node;
- The LE set of candidate super points from each observation node to the global server.

In the above data, the size of REC is fixed. The size of candidate super points and LE in transmission depends on the number of candidate super points. From the running process of READ, it can be seen that the candidate super points generated by READ when running in a single node environment are the same as those generated when running in a distributed environment. Therefore, the number of candidate super points generated at runtime under a single node can be used to determine the size of data transmission between observation nodes and the global server in a distributed environment.

Table 4 lists data transmission between each observation node and the global server. The number of candidate super points is the number of candidate super points produced by REC. The size of candidate super points is multiplied by 4 bytes (each IPv4 address size is 4 bytes); the size of candidate super points' LE is multiplied by 2^{11} bytes (LE contains 2^{14} bits, 2^{11} bytes). The total amount of data transmitted is the sum of the size of REC, the size of candidate super point and the size of LE of candidate super points. The master data structure size is the sum of REC and LEV. The percentage of transmitted data is the ratio of the total amount of transmitted data to the size of the master data structure. From Table 4, it can be seen that the average amount of data transmitted by READ between the global server and each observation node is not more than 7.5 MB, which only occupies less than 2.3% of the total size of master data structure.

Table 4. Transmitting data between each observation node and the global server.

Experiment Traffic	Statistic Name	Number of Candidate Super Points	Size of REC(MB)	Size of Candidate Super Points(MB)	Size of Candidate Super Points' LE(MB)	Total Transmission(MB)	Sum Size of REC and LEA(MB)	Pecentage of Transmission(%)
Caida 2015_02_19	Average	955.3333	3	0.00364	1.86589	4.86953	323	1.50759
	Min	801	3	0.00306	1.56445	4.56751	323	1.41409
	Max	1106	3	0.00422	2.16016	5.16438	323	1.59888
	Std	83.9224	0	0.00032	0.16391	0.16423	0	0.05085
Caida 2016_01_21	Average	363.66667	3	0.00139	0.71029	3.71167	323	1.14912
	Min	303	3	0.00116	0.5918	3.59295	323	1.11237
	Max	404	3	0.00154	0.78906	3.7906	323	1.17356
	Std	31.00831	0	0.00012	0.06056	0.06068	0	0.01879
IPtas 2017_10_23	Average	2199.1667	3	0.00839	4.29525	7.30364	323	2.26119
	Min	1723	3	0.00657	3.36523	6.37181	323	1.9727
	Max	3434	3	0.0131	6.70703	9.72013	323	3.00933
	Std	494.0519	0	0.00188	0.96495	0.96683	0	0.29933
IPtas 2018_03_08	Average	2254.9167	3	0.0086	4.40413	7.41274	323	2.29496
	Min	1790	3	0.00683	3.49609	6.50292	323	2.01329
	Max	3753	3	0.01432	7.33008	10.34439	323	3.2026
	Std	555.1954	0	0.00212	1.08437	1.08648	0	0.33637

7. Discussion

From the experimental results, it can be seen that for the network with only one observation node, the memory consumption and the estimation accuracy of READ are similar to that of the existing algorithms. This is because both READ and the existing algorithms estimate the cardinalities based on LE. However, in the distributed environment with multiple observation nodes, the communication overhead of READ is much lower than that of other algorithms. This is because READ does not need to transmit all the data structures used to estimate the cardinalities in the distributed environment, thus reducing the communication between observation nodes and the global server. In addition, READ processes each IP packet with the time complexity of O(1), and has no read-write conflict. Hence, READ can perform fast calculation on the parallel environment, so as to realize real-time super points detection in high-speed network.

From the above discussion, the following conclusions can be drawn:

- The memory consumption and error rate of READ is similar to the existing algorithms.
- The running time of READ is small enough to handle 40Gb/s networks in real time.
- In a distributed environment, READ only needs to transmit up to 10.4 MB of memory between each observation node and the global server, which accounts for less than 3.21% of the size of master data structure. It is obviously superior to other algorithms and has the advantage of low communication overhead.

8. Conclusions

READ uses REC to generate candidate super points in a distributed environment. REC is a three-dimensional structure of RE. Because RE has the characteristics of small memory occupation and fast computing speed, REC can generate candidate super points from 40Gb/s high-speed network with only 3MB of memory. LEA is used to estimate the cardinalities of candidate super points and filter out the super points. READ does not need to transfer the entire LEA to the global server. For 40 Gb/s high-speed network, the data size transmitted between each observation node and the global server is only 3.21% of the sum of REC and LEA. Low data communication overhead ensures the efficient operation of READ in a distributed environment even under the sliding time window. READ can realize super points detection in a distributed environment. However, the detected super points may be normal servers, scanners, P2P nodes, or even dark network routing nodes. Future research will focus on classifying these super points in the distributed environment and detecting suspicious or malicious super points in the distributed environment.

Author Contributions: Conceptualization, J.X. and W.D.; methodology, J.X.; software, J.X.; validation, J.X. and W.D.; formal analysis, J.X. and W.D.; investigation, J.X. and W.D.; resources, X.X.; data curation, J.X. and W.D.; writing—original draft preparation, J.X.; writing—review and editing, J.X. and W.D.; visualization, J.X.; supervision, J.X.; project administration, J.X. and W.D.; funding acquisition, J.X. and W.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the project of Jiangsu Provincial Department of Education OF FUNDER grant number 20KJB413002; the science and technology research project of Jiangsu Provincial Public Security Department OF FUNDER grant number 2020KX007Z; the Jiangsu Police Institute high level talent introduction research start-up fund (JSPIGKZ)grant number JSPI20GKZL404.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The network traffic used in this paper could be acquired from CAIDA "http://www.caida.org/data/passive (accessed on 24 September 2021)" and IPtas "http://iptas.edu.cn/src/system.php (accessed on 24 September 2021)".

Conflicts of Interest: The authors declare no conflict of interest.

References

1. China Internet Network Information Center (CNNIC). *China Internet Network Development Statistic Report*, 43th; China Internet Network Information Center (CNNIC): Beijing, China, 2019.
2. Ai-ping, Z. Research on the Key Issues of Traffic Measurement in High-Speed Networks. Ph.D. Thesis, Southeast University, Nanjing, China, 2015.
3. Kucera, J.; Kekely, L.; Piecek, A.; Korenek, J. General IDS Acceleration for High-Speed Networks. In Proceedings of the 2018 IEEE 36th International Conference on Computer Design (ICCD), Orlando, FL, USA, 7–10 October 2018; pp. 366–373. doi:10.1109/ICCD.2018.00062.
4. Venkataraman, S.; Song, D.; Gibbons, P.B.; Blum, A. New Streaming Algorithms for Fast Detection of Superspreaders. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–27 February 2005; pp. 149–166.
5. Modi, C.; Patel, D.; Borisaniya, B.; Patel, H.; Patel, A.; Rajarajan, M. A survey of intrusion detection techniques in Cloud. *J. Netw. Comput. Appl.* **2013**, *36*, 42–57. doi:10.1016/j.jnca.2012.05.003.

6. Kamiyama, N.; Mori, T.; Kawahara, R. Simple and Adaptive Identification of Superspreaders by Flow Sampling. In Proceedings of the IEEE INFOCOM 2007—26th IEEE International Conference on Computer Communications, Anchorage, AK, USA, 6–12 May 2007; pp. 2481–2485. doi:10.1109/INFCOM.2007.305.
7. Wang, P.; Guan, X.; Qin, T.; Huang, Q. A Data Streaming Method for Monitoring Host Connection Degrees of High-Speed Links. *IEEE Trans. Inf. Forensics Secur.* **2011**, *6*, 1086–1098. doi:10.1109/TIFS.2011.2123094.
8. Liu, W.; Qu, W.; Gong, J.; Li, K. Detection of Superpoints Using a Vector Bloom Filter. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 514–527. doi:10.1109/TIFS.2015.2503269.
9. Yoon, M.; Li, T.; Chen, S.; Peir, J.K. Fit a Compact Spread Estimator in Small High-speed Memory. *IEEE/ACM Trans. Netw.* **2011**, *19*, 1253–1264. doi:10.1109/TNET.2010.2080285.
10. Xu, J.; Ding, W.; Hu, X. Most Memory Efficient Distributed Super Points Detection on Core Networks. In *Algorithms and Architectures for Parallel Processing*; Vaidya, J., Li, J., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 153–167.
11. Xu, Y.; Wang, G.; Ren, J.; Zhang, Y. An adaptive and configurable protection framework against android privilege escalation threats. *Future Gener. Comput. Syst.* **2019**, *92*, 210–224. doi:10.1016/j.future.2018.09.042.
12. Cheng, G.; Tang, Y. Line speed accurate superspreader identification using dynamic error compensation. *Comput. Commun.* **2013**, *36*, 1460–1470. doi:10.1016/j.comcom.2013.05.006.
13. Liu, Z.; Wang, R.; Tao, M.; Cai, X. A class-oriented feature selection approach for multi-class imbalanced network traffic datasets based on local and global metrics fusion. *Neurocomputing* **2015**, *168*, 365–381. doi:10.1016/j.neucom.2015.05.089.
14. Zheng, Y.; Li, M. Towards More Efficient Cardinality Estimation for Large-Scale RFID Systems. *IEEE/ACM Trans. Netw.* **2014**, *22*, 1886–1896. doi:10.1109/TNET.2013.2288352.
15. Adam, H.; Yanmaz, E.; Bettstetter, C. Contention-Based Estimation of Neighbor Cardinality. *IEEE Trans. Mob. Comput.* **2013**, *12*, 542–555. doi:10.1109/TMC.2012.19.
16. Li, B.; He, Y.; Liu, W. Towards Constant-Time Cardinality Estimation for Large-Scale RFID Systems. In Proceedings of the 2015 44th International Conference on Parallel Processing, Beijing, China, 1–4 September 2015; pp. 809–818. doi:10.1109/ICPP.2015.90.
17. Flajolet, P.; Martin, G.N. Probabilistic counting. In Proceedings of the 24th Annual Symposium on Foundations of Computer Science (sfcs 1983), Tucson, AZ, USA, 7–9 November 1983; pp. 76–82. doi:10.1109/SFCS.1983.46.
18. Flajolet, P.; Fusy, E.; Gandouet, O.; Meunier, F. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In Proceedings of the Analysis of Algorithms 2007 (AofA07), Juan les Pins, France, 17–22 June 2007; pp. 127–146.
19. Whang, K.Y.; Vander-Zanden, B.T.; Taylor, H.M. A Linear-time Probabilistic Counting Algorithm for Database Applications. *ACM Trans. Database Syst.* **1990**, *15*, 208–229. doi:10.1145/78922.78925.
20. Xu, J.; Ding, W.; Gong, J.; Hu, X.; Liu, J. High Speed Network Super Points Detection Based on Sliding Time Window by GPU. In Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, 12–15 December 2017; pp. 566–573. doi:10.1109/ISPA/IUCC.2017.00092.
21. Xu, J.; Ding, W.; Gong, J.; Hu, X.; Sun, S. *SRLA*: A Real Time Sliding Time Window Super Point Cardinality Estimation Algorithm for High Speed Network Based on GPU. In Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, 28–30 June 2018; pp. 942–947. doi:10.1109/HPCC/SmartCity/DSS.2018.00156.
22. Xu, J.; Ding, W.; Gong, Q.; Hu, X.; Yu, H. A Super Point Detection Algorithm Under Sliding Time Windows Based on Rough and Linear Estimators. *IEEE Access* **2019**, *7*, 43414–43427. doi:10.1109/ACCESS.2019.2908226.
23. Coskun, B. (Un)wisdom of Crowds: Accurately Spotting Malicious IP Clusters Using Not-So-Accurate IP Blacklists. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1406–1417. doi:10.1109/TIFS.2017.2663333.
24. Cianfrani, A.; Eramo, V.; Listanti, M.; Polverini, M.; Vasilakos, A.V. An OSPF-Integrated Routing Strategy for QoS-Aware Energy Saving in IP Backbone Networks. *IEEE Trans. Netw. Serv. Manag.* **2012**, *9*, 254–267. doi:10.1109/TNSM.2012.031512.110165.
25. Xiao, L.; Xia, X.G. A new robust Chinese remainder theorem with improved performance in frequency estimation from undersampled waveforms. *Signal Process.* **2015**, *117*, 242–246. doi:10.1016/j.sigpro.2015.05.017.
26. Christensen, K.; Roginsky, A.; Jimeno, M. A new analysis of the false positive rate of a Bloom filter. *Inf. Process. Lett.* **2010**, *110*, 944–949. doi:10.1016/j.ipl.2010.07.024.
27. Xu, J.; Ding, W.; Hu, X.; Gong, Q. VATE: A trade-off between memory and preserving time for high accurate cardinality estimation under sliding time window. *Comput. Commun.* **2019**, *138*, 20–31. doi:10.1016/j.comcom.2019.02.005.
28. CAIDA. The CAIDA Anonymized Internet Traces. Available online: http://www.caida.org/data/passive (accessed on 24 September 2021).
29. IPtas. Network Technology Key Labratory of Jiangsu Province, IP Trace And Service. Available online: http://iptas.edu.cn/src/system.php (accessed on 24 September 2021)