IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 26, 2019, accepted March 18, 2019, date of publication March 29, 2019, date of current version April 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2908226

A Super Point Detection Algorithm Under Sliding Time Windows Based on Rough and Linear Estimators

JIE XU[1], WEI DING[2], QIUSHI GONG[2], XIAOYAN HU[2], (Member, IEEE), AND HAIQING YU[2]
[1]School of Computer Science and Engineering, Southeast University, Nanjing 211189, China
[2]School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China

Corresponding author: Jie Xu (xujieip@163.com)

This work was supported by the National Natural Science Foundation of China under Grant 61602114.

ABSTRACT Detecting super points from high-speed networks can effectively help to monitor networks, which is a hot topic in network fields. Most existing algorithms are carried out under discrete time windows and the results are in a certain percentage of omission. In this paper, the phenomenon of missed super points detection in discrete time windows is analyzed based on real-world traffic. Then a new algorithm, which detects the super points under sliding time windows, is proposed. Our algorithm uses a lightweight estimator to identify candidate super points and a linear estimator to filter super points. The lightweight estimator is fast, and the linear estimator has high accuracy. Both the lightweight estimator and the linear estimator adopt a data structure, called distance recorder, to support sliding time windows. Moreover, our algorithm is also a parallel algorithm. On the basis of thoroughly discussing the mathematic principles and operation steps of our algorithm, two groups of real-world traffic from a 40-Gb/s high-speed network are applied in the experiments which running on a graphic processing unit (GPU). The experiments are conducted under discrete time windows and sliding time windows separately. The former results show that our algorithm is superior to other existing algorithms in the comprehensive performance, and the latter results indicate that our algorithm can run steadily under sliding time windows.

INDEX TERMS Cardinality estimation, GPU, sliding time window, super points detection.

I. INTRODUCTION

Attribute measurement is the basis of network management and security [1]. Suppose there are two core networks, denoted as *ANet* and *BNet*. *ANet* and *BNet* communicate with each other through an edge router (*ER*). For a host "*aip*" in *ANet*, the cardinality of *aip* is the number of hosts in *BNet* communicating with it in a certain period [2]. Host cardinality is an important attribute in the field of network management and measurement [3]. In a specified period, if the cardinality of *aip* is no smaller than a predefined threshold θ, *aip* is called a super point [4]. Furthermore, this period is called a time window.

Super point is a kind of particular host which is associated with many network events, such as network servers [5], Distributed Denial of Service (DDoS) [6], scanning [7] etc. Because super points only account for a small part of the total hosts, monitoring super points is an efficient way to manage the large-scale network [8]. Super points detection is a hot research field and many algorithms [9], [10], [11] are proposed in recent years.

A time window could be a discrete time window or a sliding time window [12]. Under sliding time windows, the detection result is more accurate and the delay is smaller than that under discrete time windows. However, the existing algorithms [9], [10], [11] can not run under sliding time windows, because they can not save the state of the previous period. Moreover, the existing algorithms only detect the super point at the end of the time window and take a long time for the detection. Due to the continual super point detection under sliding time windows, the existing algorithms also fail to meet the processing time requirement.

To solve the above problems, this paper presents a sliding time window super point detection algorithm, named as sliding rough and linear algorithm (*SRLA*). *SRLA* can preserve the host state of previous period. Moreover, *SRLA* takes less

The associate editor coordinating the review of this manuscript and approving it for publication was Kuan Zhang.

than 0.6% of super point cardinality estimation time of other algorithms. Because under the sliding time window, super point detection is more frequent than that under the discrete time window, the small super point cardinality estimation time of *SRLA* ensures the success running under sliding time windows in real time.

The advantages of *SRLA* come from two novel estimators proposed in this paper: sliding rough estimator (*SRE*) and sliding linear estimator (*SLE*). Both *SRE* and *SLE* use distance recorder (*DR*), a counter that can record the state of the host under the sliding time window, to save the state of the host. So *SRLA* can run under sliding time windows. *SRE* and *SLE* have different speed and accuracy. *SRE* is a lightweight algorithm. It occupies less memory and runs fast. Therefore, *SRE* can determine whether a host is a super point when scanning a network packet. *SRLA* uses *SRE* to generate a candidate super point list when scanning network packets, which significantly reduces the super point detection time. *SLE* estimates the cardinality of a given host accurately. At the end of a time window, *SRLA* uses *SLE* to estimate the cardinality of candidate super point and removes those whose estimated cardinalities are lower than the threshold. *SLE* guarantees the high accuracy of *SRLA*. Because of the fast detection speed of *SRE* and the high accuracy of *SLE*, the speed and accuracy of *SRLA* are better than the previous algorithms. What's more, *SRLA* can run under sliding time windows.

The development of computing platform, especial graphic process unit (GPU), provides a better environment [13] for super point cardinality estimation under sliding time windows. This paper makes full use of the parallel computing ability of GPU to handle high-speed network traffic in real time. Some of the work in this paper has been published at the 20th International Conferences on High Performance Computing and Communications in 2018 [14].

The main contributions of this paper are listed below:

1) Devise a lightweight estimator and a high accuracy estimator. These two estimators can maintain the state of previous period and run under the sliding time window.
2) Propose a super point cardinality estimation algorithm under the sliding time window, which uses a fixed quantity of the lightweight estimator and the high accuracy estimator. This algorithm has the fastest super point detection speed and is the only one which can run under the sliding time window.
3) Deploy super point detection algorithms on GPU to handle high-speed network traffic parallel and evaluate the performance of these algorithms on a real-world 40 Gigabits per second (Gbps) network under the discrete time window and the sliding time window respectively.

In section II, we introduce the related works. In section III, two novel sliding time window cardinality estimators are described in detail. Section IV demonstrates *SRLA* in detail. How to deploy *SRLA* on GPU and numerical results on a real-world 40 Gbps network are shown in section V. The paper is concluded in the final section.

II. RELATED WORK

For convenience, suppose the input data is the stream of IP pair tuples like $< aip, bip >$ where *aip* is a host in *ANet* and *bip* is a host in *BNet*.

The statistical method, which consumes a large and variable amount of memory, stores every distinct opposite host to calculate *aip*'s cardinality [15]. The accurate result provides a baseline to measure the performance of other algorithms. But the inefficiency of this method prevents it from being deployed for real-time network monitor [16],

Estimating-based methods [17] are proposed to improve the processing speed and reduce the memory consumption. These methods are based on cardinality estimator.

Whang *et al.* [18] proposes a classical algorithm, Linear Estimator (*LE*), to estimate the number of distinct elements (the cardinality of hosts) from a data stream (IP pair stream). *LE* contains g' bits which are initialized to 0 at the beginning. It randomly hashes [19] every element to a bit and sets the bit to 1. After scanning all elements in a time window, *LE* estimates the distinct elements number according to equation (1). In equation (1), n_0 is the number of '0' bits. The mathematical theorem of *LE* has been described in detail in [18].

$$Est = -g' * ln(\frac{n_0}{g'}) \tag{1}$$

Calculating the cardinality of *aip* is to count its distinct opposite hosts' number. Hence *LE* is an excellent method to detect super points.

Since large quantity of hosts may appear in a time window, assigning *LE* for every host will be inefficient [20]. Therefore, the detection of all super points and the estimation of their cardinalities are hot research topics. Several representative algorithms have been proposed [9], [10], [11], and we refer to this kind of algorithms as estimating algorithm. Estimating algorithm uses a fixed length of memory to estimate host cardinality, and contains three procedures:

1) Scan every IP pair and record cardinality information in a special data structure;
2) Detect super points and give a candidate list;
3) Estimate the cardinality of every host in the candidate list and remove those whose cardinalities are smaller than the threshold.

All these algorithms' primary structures consist of several *LE*. The differences of these algorithms are how to map *aip* to *LE* and how to identify super points.

Wang *et al.* [9] proposes double connection degree sketch (*DCDS*) to detect super point based on the Chinese remainder theorem (*CRT*) [21]. *DCDS* contains an array of *LE*. Each host is mapped to different *LE* and the super point is restored by *CRT*. But the *CRT* contains complex operations and limits the speed of *DCDS*. Liu *et al.* [10] proposes vector bloom filter algorithm (*VBFA*) to detect super point according to the principle of bloom filter. *VBFA* maps every host to

different *LE* by extracting sub bits from the IP address of the host and restores super point by combining sub bits of IP address. There are no complex operation in *VBFA*. Hence *VBFA* scans packets quickly.

Unlike *DCDS* and *VBFA*, the grand spread estimator (*GSE*), proposes by Shin *et al.* [11], uses a fixed length bit pool as the primary structure. Every *aip* has a virtual estimator with g' bits extracted from the bit pool. The virtual estimator is updated like *LE*. At the end of a time window, *aip*'s cardinality is estimated according to the number of zero bits in its virtual estimator and the fraction of zero bits in the bit pool. The accuracy of this algorithm is affected by the length of bit pool and the number of distinct IP pairs in a time window. *GSE* is the first algorithm which uses GPU [22] to estimate cardinalities.

All of these super point detection algorithms cannot work under the sliding time window. The primary reason is that they cannot record the appearance time of different hosts. The second reason is that they spend too much time on super points reconstruction to work in real time under the sliding time window. *SRLA* proposed in this paper will solve these two problems.

III. CARDINALITY ESTIMATOR UNDER SLIDING TIME WINDOWS

LE is widely used by super point detection algorithms because of its high estimation accuracy [23]. The time complexity of the cardinality estimation is $O(g')$. Paper [18] also shows that the accuracy of *LE* is related to g', that is, the larger the g' is, the higher the accuracy of *LE* will be.

Since the super point is only a small fraction of all hosts [24], there is no need to evaluate the cardinality of every host that appears. It is a common train of thought for super point detection algorithms to select super point without estimate every host's cardinality [25]. Existing algorithms use *LE* to detect super points and estimate their cardinalities at the same time. In the practice of super point detection, *LE* cannot meet the accuracy and real-time requirements simultaneously because of its high operational complexity [26]. Additionally, since all these algorithms cannot preserve previous information, they cannot be adopted under sliding time windows.

In this section, we will try to solve this problem with a rough estimator derived from an optimal estimating algorithm [27]. By modifying the rough estimator, we can find these hosts which may exceed the threshold effectively. At the end of a time window, only cardinalities of candidate super points found by rough estimator will be estimated by a high accuracy estimator, linear estimator. By introducing a novel structure, Distance Recorder (*DR*), the sliding rough estimator (*SRE*) and the sliding linear estimator (*SLE*) are proposed to detect and estimate super points' cardinalities under sliding time windows. *SRE* detects candidate super points quickly by scanning each IP pair $< aip, bip >$, and *SLE* estimates the cardinalities of these candidate super points with high accuracy at the end of each time slice. Time slice is the basic unit of time in this paper. A time window consists of one or several time slices. In the latter section, we propose an algorithm that can simultaneously estimate several hosts' cardinalities and detect candidate super points with a fixed amount of memory based on *SRE* and *SLE* proposed in this section.

Estimating the cardinality of a single *aip* is the basis for super point detection and cardinality estimation. This section will describe the procedure of estimating the cardinality of a single host under the sliding time window.

In the first place, we discuss the difference between the discrete time window and the sliding time window.

A. DISCRETE TIME WINDOW AND SLIDING TIME WINDOW

A time window moves forward one time slice a time. A discrete time window consists of only one time slice. But a sliding time window may contain several time slices. FIGURE 1 demonstrates their difference by an example. In FIGURE 1, the size of a time window is 300 seconds and the observing period is 600 seconds. Under discrete time windows, there are only two time windows: *DW1* and *DW2*. On the other hand, the sliding time window moves forward one second a time, there are 301 sliding time windows, from *SW1* to *SW301*. Obviously, detection result under the sliding time window is better than that under discrete time windows at the cost of additional estimation procedures. Let $W(t, k)$ represent a time window containing k continuous time slices which starting from time slice t. The last time slice of $W(t, k)$, i.e. time slice $t + k - 1$, is the current time slice.

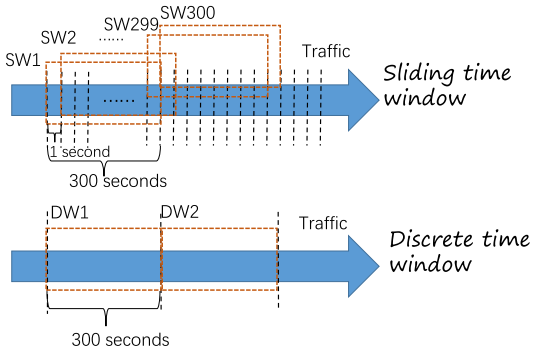


FIGURE 1. Sliding time windows and discrete time windows.

Suppose there is a data stream, denoted by ST. $ST =< bip_1, bip_2, bip_3, bip_4, \cdots, bip_i, \cdots >$, where bip_i is an IP address in *BNet*. For a host "*aip*" in *ANet*, the ST here is the data stream composed of its opposite hosts. Each *bip* in ST belongs to a time slice. Let $ST(t, k)$ represent the sub stream of ST in the time window $W(t, k)$. The cardinality of $ST(t, k)$ is the number of distinct *bip* in $ST(t, k)$ and is represented by $|ST(t, k)|$.

There are the following two problems under discrete time windows:

1) False negative. Affected by the starting point of the time window, algorithm under discrete time windows can't

detect super points which span the boundary of two time windows [28].

2) Detection latency. Algorithm under discrete time windows won't detect super points until the end of the time window [29].

Both of these weaknesses come from the moving step of the discrete time window [30]. The larger the size of a discrete time window is, the more false negative rate and higher detection latency will occur. Therefore, we use the sliding time window to cope with these problems by adopting fine-grained moving steps [31]: the size of steps is sufficiently small so that the process can be viewed as a continuously sliding time window as suggested by its name. But detecting super points under the sliding time window is more difficult considering the intense observing frequency [32]. What's more, to work under the sliding time window, an algorithm must preserve hosts' state of a previous period [33].

To measure the loss rate of the discrete time window, we captured a one-hour traffic at the edge of Nanjing main node (POP Nanjing) of China Education and Research Network (CERNET) from 13:00 to 14:00, Mar. 8th, 2018 [34]. This traffic is also one of the experiment data and its detailed feature is listed in section V.

TABLE 1. Statistic result of discrete time windows vs. sliding time windows.

Start time	End time	DWSP	SWSP	SWSP - DWSP	FalseRatio(%)
13:00	13:10	549	574	25	4.355401
13:05	13:15	558	579	21	3.626943
13:10	13:20	567	583	16	2.744425
13:15	13:25	578	594	16	2.693603
13:20	13:30	581	598	17	2.842809
13:25	13:35	581	595	14	2.352941
13:30	13:40	579	598	19	3.177258
13:35	13:45	585	603	18	2.985075
13:40	13:50	584	602	18	2.990033
13:45	13:55	580	596	16	2.684564
13:50	14:00	610	627	17	2.711324

* FalseRatio(%)=100 * (SWSP-DWSP)/SWSP

The statistical results of 10 minutes observing period, under discrete time windows and sliding time windows as shown in FIGURE 1, are listed in TABLE 1. The threshold θ is set as 1024. Data in column "*DWSP*" are the size of the union of the super points detected from two discrete time windows (*DW1* and *DW2* in FIGURE 1). Data in column "SWSP" are the size of the union of the super points detected from 301 sliding time windows (*SW1* to *SW301* in FIGURE 1). As shown in the table, more than 14 super points are lost in every observing period. Hence, it is necessary to adopt sliding time windows for high accuracy [35]. In the following part we will introduce a rough estimator under discrete time windows and sliding time windows respectively.

B. ROUGH ESTIMATOR

Kane *et al.* [27] presents a general rough estimation algorithm, denoted as RE_0, to estimate the cardinality of any data stream. By applying this algorithm, $|ST(t,1)|$ can be roughly estimated using only g integers (for IP address of version 4, $g = 8$, and each integer is 5 bits). RE_0 uses g integers to record the least significant bits of hashed elements in the data stream. Let REI represent these g integers. $REI[i]$ represents the i-th integer of REI. Each integer in REI is initialized to 0. The definition of the least significant bit used in RE_0 is provided below:

Definition 1 (Least Significant Bit): Given an integer i, $BIT(i)$ represents the binary format of i. The least significant bit of i is the position the first bit '1' in $BIT(i)$ starting from the right, denoted by $LSB(i)$.

For example, $LSB(3) = 0$, $LSB(40) = 3$, because the binary numbers of 3 and 40 are "11" and "101000" respectively. Let $H(x, n, A)$ represent a hash function that can randomly map an integer x to a number between 0 and $n-1$ according to random seed parameter A [19], where n is an positive integer. For each bip in $ST(t,1)$, RE_0 maps "bip" randomly to an integer between 0 and $2^{32}-1$ using the hash function $H(bip, 2^{32}, A_0)$ with random seed parameter A_0. The hash function $H(bip, g, A_1)$ with random seed parameter A_1 is used to map "bip" to an integer of REI, i.e. $REI[H(bip, g, A_1)]$. $REI[H(bip, g, A_1)]$ is used to hold the maximum least significant bit of $H(bip, 2^{32}, A_0)$, that is, $REI[H(bip, g, A_1)] = max(REI[H(bip, g, A_1)], LSB(H(bip, 2^{32}, A_0)))$. After scanning all the elements in a time window, RE_0 estimates the cardinality based on REI. Let $T(r')$ indicate the number of elements in REI whose value are no smaller than r' (r' is an integer). If $T(r') \geq g * \rho$ and $T(r'+1) < g * \rho$, the estimating cardinality, denoted by C', is given by formula (2), where $\rho = 0.99 * (1 - e^{-\frac{1}{3}})$.

$$C' = 2^{r'} * g \tag{2}$$

The lemma 4 of paper [27] proves that the probability of C' belonging to $[|ST(t,1)|, 4*|ST(t,1)|]$ is $1 - O(\frac{1}{g})$. RE_0 needs to record the least significant bits of each hashed element. But in the super point detection process, we only need to determine whether the cardinality is higher than the specified threshold θ. A threshold of the lowest significant bit, denoted by τ, can be determined according to θ. In equation (2), τ is the value of r' when C' is the threshold θ. Equation (3) shows how to calculate τ according to θ.

$$\tau = log_2(\frac{\theta}{g}) \tag{3}$$

Only hashed elements whose least significant bit are no smaller than τ should be recorded. The rough estimator proposed in this paper, denoted as RE, is based on this idea. RE uses g bits instead of g integers of RE_0. Hence REI in RE_0 becomes a string of bits with length g, denoted as rough estimator bits (REB). Every bit in REB is initialized to 0. Let $REB[i]$ indicate the i-th bit in it. The weight of REB, denoted by $|RE|$, represents the number of 1 bits in REB. Therefore, RE can be used to determine whether the number of distinct elements in the data stream $ST(t,1)$ is higher than the threshold. It contains two procedures: updating REB by bip and checking if aip's cardinality is bigger than threshold

by *REB*. The Boolean function description of *RE* is given below.

RE_Update(*bip*):

1) Hash *bip* to a random value b' between 0 and $2^{32}-1$ by $H(bip, 2^{32}, A_0)$;
2) Calculate the least significant bit of b', $LSB(b')$;
3) If $LSB(b')$ is smaller than τ, return.
4) Hash *bip* to an integer i by $H(bip, g, A_1)$;
5) Set the i-th bit of *REB*, $REB[i]$;

RE_IsSP() :check if *aip*'s cardinality is bigger than θ;

1) Get the number of bit '1' in *REB*, store it to n_1;
2) If n_1 is no smaller than $\rho * g$, return TRUE, else return FALSE.

If an *aip*'s cardinality is no smaller than θ, *RE_IsSP*() detects it out with high probability. The theorems that support *RE* are provided below.

Lemma 1: Assumes that there are α different balls and g different boxes ($\alpha \geq g$). Throw all the balls into these boxes randomly. Let $AN(\alpha, g)$ indicate that there is at least one ball in each box. Then $AN(\alpha, g) = g^\alpha - \sum_{i=1}^{g-1} * C_g^i * AN(\alpha, i)$ and $AN(\alpha, 1) = 1$.

Proof: There are total g^α different cases throwing balls. When there is only one box, there is only one way to throw balls, throwing all the balls into the single box. When just i boxes are none empty, there are $C_g^i * AN(\alpha, i)$ cases, and $1 \leq i \leq g$. Subtract these cases that at least one box is empty from $g^{\hat{1}\acute{s}}$. And we will get the cases that all boxes are none empty, that is, $AN(\alpha, g) = g^\alpha - \sum_{i=1}^{g-1} * C_g^i * AN(\alpha, i)$. □

Theorem 1: Throw α balls into g boxes. Let g_1 represent the number of none empty boxes and $FN(\alpha, g, g_1)$ represent the number of cases that there are g_1 none empty boxes in g boxes. Then $FN(\alpha, g, g_1) = C_g^{g_1} * AN(\alpha, g_1)$ and $1 \leq g_1 \leq g$.

Proof: The remaining $g - g_1$ boxes are empty. Choose $g - g_1$ empty boxes from g boxes. There are $C_g^{g_1}$ selection methods. Each case has $AN(\alpha, g_1)$ different methods to throw these balls. So the total situation is $C_g^{g_1} * AN(\alpha, g_1)$. □

$ST(t, 1)$ can be regarded as a set of balls, and *REB* is treated as a set of boxes in theorem 1. Since *RE* uses a hash function to sample data in $ST(t, 1)$, not every element in $ST(t, 1)$ updates a bit in *REB*. Suppose that there exist α different elements in $ST(t, 1)$ updating *REB*. The probability of $|RE| = g_1$, denoted by $\mathbb{P}_w\{\alpha, g, g_1\}$, is given in equation (4).

$$\mathbb{P}_w\{\alpha, g, g_1\} = \frac{FN(\alpha, g, g_1)}{g^\alpha} \tag{4}$$

Each element in $ST(t, 1)$ has the probability of $\frac{1}{2^\tau}$ updating *REB*. Let $\mathbb{P}_{bip}\{|ST(t, 1)|, \tau, \alpha\}$ represent the probability that there are α different elements in $ST(t, 1)$ updating *REB*. Equation (5) shows how to calculate $\mathbb{P}_{bip}\{|ST(t, 1)|, \tau, \alpha\}$ from $|ST(t, 1)|$.

$$\mathbb{P}_{bip}\{|ST(t, 1)|, \tau, \alpha\} = C^\alpha_{|ST(t,1)|} * (\frac{1}{2^\tau})^\alpha * (1 - \frac{1}{2^\tau})^{|ST(t,1)|-\alpha} \tag{5}$$

Combining equations (4) and (5), we can get the probability of there are g_1 bits of *REB* being '1' after scanning $ST(t, 1)$, denoted by $\mathbb{P}_s\{|ST(t, 1)|, g, \tau, g_1\}$. Equation (6) shows how to calculate $\mathbb{P}_s\{|ST(t, 1)|, g, \tau, g_1\}$.

$$\mathbb{P}_s\{|ST(t, 1)|, g, \tau, g_1\} = \sum_{\alpha=g_1}^{|ST(t,1)|} \mathbb{P}_{bip}\{|ST(t, 1)|, \tau, \alpha\} * \mathbb{P}_w\{\alpha, g, g_1\} \tag{6}$$

The probability that at least n bits are set in window $W(t, 1)$, denoted by $\mathbb{P}_s^+\{|ST(t, 1)|, g, \tau, n\}$, can be derived from formula (6), as shown in equation (7).

$$\mathbb{P}_s^+\{|ST(t, 1)|, g, \tau, n\} = \sum_{g_1=n}^{g} \mathbb{P}_s\{|ST(t, 1)|, g, \tau, g_1\} \tag{7}$$

Assuming that $|ST(t, 1)| = \theta = 1024$, then the number of different elements in $ST(t, 1)$ updating *REB* is expected to be 8. When g=8, a super point is identified by *RE* with probability more than 99.9576% according to equation (7). Comparing with RE_0 [27], *RE* occupies one-fifth of RE_0's memory. *RE* updates *REB* for only about $\frac{1}{127}$ elements in $ST(t, 1)$ when $\theta = 1024$ with memory access operations of only $\frac{1}{127}$ of RE_0. However, *RE* only works under the discrete time window. The sliding rough estimator to be described in the next section enables *RE* to work under the sliding time window by adopting a powerful counter.

C. SLIDING ROUGH ESTIMATOR

The rough estimator *RE* can estimate whether the number of distinct elements in a window exceeds the threshold. This section makes the corresponding adjustment so that it can be applied under the sliding time window. The adjusted estimator is called *Sliding Rough Estimator* (*SRE*).

Any two adjacent sliding time windows with k time slices have $k-1$ overlapping time slices. For an element *bip* in the data stream, it may appear multiple times in different time slices. How to judge if *bip* appears in the current time window is the problem that must be faced under the sliding time window. In this paper we use a data structure called *Distance Recorder* (*DR*) to solve this problem.

DR is a counter consisting of z bits and it records the distance between the time slice of the last occurrence of *bip* and the current time slice, marked as *dr(bip)*. And consequently *dr(bip)* being 0 suggests that *bip* appears in the current time slice. Each time the window slides, the "+1" operation is performed on *DR*. When $dr(bip) \in [0, k-1]$ (less than k), *bip* appears in the sliding time window $W(t, k)$. When $dr(bip) = 2^z - 1$, the distance of *bip* is more than k from the current time slice, that is, *bip* does not appear in the current window. k is the number of time slices within the sliding time window, with a maximum value no bigger than $2^z - 1$. When $z = 1$, then $k = 1$ and the sliding time window becomes a discrete time window, that is, *DR* can also be used for discrete time windows.

The initial value of the $dr(bip)$ is $2^z - 1$. *DR* has four main operations listed as follows (bip_1 and bip_2 represent two different elements in the data stream):

1) $DRinit(dr(bip_1))$: set every bit of $dr(bip_1)$ to 1;
2) $DRset(dr(bip_1))$: set every bit of $dr(bip_1)$ to 0;
3) $DRslide(dr(bip_1))$: preserve the value of $dr(bip_1)$ when the time window sliding; If the value of $dr(bip_1)$ is smaller than $2^z - 1$, increase $dr(bip_1)$ by 1;
4) $DRjoin(dr(bip_1), dr(bip_2))$: return a new *DR* whose value is the maximum of $dr(bip_1)$ and $dr(bip_2)$;

$DRinit(dr(bip_1))$ is used to initialize $dr(bip_1)$ at the beginning of the algorithm; $DRset(dr(bip_1))$ is used to process elements that appear in the current time slice; $DRslide(dr(bip_1))$ updates the value of $dr(bip_1)$ when the window slides; multiple *DR* can be combined into a new *DR* using the *DRjoin* operation.

The above operations of *DR* guarantee that *DR* can maintain the correct distance value. The method of calculating the cardinality of $ST(t, k)$ by the statistical approach is to assign a *DR* to each distinct element. At the end of each time slice, $|ST(t, k)|$ is accurately calculated by counting the number of *DR* whose values are less than k. However, the memory and computing cost of this method is significant. For instance, when the elements in *ST* are IPv4 addresses, each element requires $32 + z$ bits: 32 bits for the IP address and z bits for the *DR*. The data stream in each time window needs $|ST(t, k)| * (32 + z)/8$ bytes. When $|ST(t, k)|$ is large, it is also a complex task to quickly locate every *DR* from the large memory. Therefore, this statistical method can only be applied to small-scale networks, or to get the standard result off-line for evaluating the accuracy of other algorithms.

In this paper, we design an estimator based on *RE*: *Sliding Rough Estimator* (*SRE*). *SRE* is a memory optimization algorithm that uses only g *DR* to determine whether $|ST(t, k)|$ exceeds a specified threshold.

SRE uses *DR* instead of the bit in *RE*. Hence the bit string *REB* becomes *DR* array, sliding rough *DR* array (*SRDR*). Let $SRDR[i]$ represent the i-th *DR* in *SRDR*. When the element of $ST(t, k)$ is IPv4 address, g is set to 8 [27].

DR preserves the time information of different elements appearing in $ST(t, k)$. It judges whether each element appears in the current time window $W(t, k)$ according to the value of *DR*. *DR* less than k corresponds to bit 1 in the *REB*. Hence the weight of the *SRE*, denoted as $|SRE|_k$, is the number of *DR* whose values less than k in the *SRDR*. All *DR* in *SRDR* are initialized to $2^z - 1$ at the beginning of the algorithm.

Let $SRE(aip)$ represent the *SRE* used by *aip*. Similar to *RE*, *SRE* includes two parts: updating *SRDR* of $SRE(aip)$ based on *bip* and determining whether *aip* exceeds the threshold based on *SRDR*. For every *bip* in a time slice, *SRE* firstly uses Algorithm 1 to check if *bip* meets the requirement of updating *SRDR*. If $SRE\_IsRecord(bip)$ returns true, then *SRE* updates *SRDR* by Algorithm 2. At the end of a time window, *SRE* determines if *aip* is a candidate super point by Algorithm 3.

SRE uses *DRset* operation when updating a *DR*. *DRset* operation sets the value of *DR* to 0, indicating that the *bip* appears in the current time slice. When the boundary of the window is reached and the super point detection is completed, all the *DR* are proceeded by the *DRslide*() function.

Algorithm 1 *SRE_IsRecord*

Input: *bip*
Output: *IsRecord*

$b' \leftarrow H(bip, 2^{32}, A_0)$
if $LSB(b') < log_2(\theta/g)$ **then**
    $IsRecord \leftarrow False$
else
    $IsRecord \leftarrow True$
end if
Return *IsRecord*

Algorithm 2 *SRE_Update*

Input: $SRE(aip)$, *bip*

$i \leftarrow H(bip, g, A_1)$
srdr $\leftarrow$ point to the *SRDR* of $SRE(aip)$
$DRset(srdr[i])$;

Algorithm 3 *SRE_IsSP*

Input: $SRE(aip)$
Output: *IsSP*

$|SRE|_k \leftarrow 0$
srdr $\leftarrow$ point to the *SRDR* of $SRE(aip)$
for $i \in [0, g-1]$ **do**
    if $srdr[i] < k$ **then**
        $|SRE|_k++$
    end if
end for
if $|SRE|_k < 0.99 * (1 - e^{-\frac{1}{3}}) * g$ **then**
    $IsSP \leftarrow False$
else
    $IsSP \leftarrow True$
end if
Return *IsSP*

SRE is simple, fast and memory economic. However, as a lightweight estimator, it can only estimate whether the cardinality is greater than the threshold in real-time when scanning the data stream. It cannot give an accurate estimation of the cardinality. For this reason, a more precise estimator is needed. This task is solved by sliding linear estimator which is introduced in the following section.

D. SLIDING LINEAR ESTIMATOR

When *SRE* determines that the value of $|ST(t, k)|$ is greater than or equal to θ, it is necessary to accurately estimate the value of $|ST(t, k)|$, which is performed by the sliding linear estimator (*SLE*) proposed in this section. By *SLE*, mistakes containing in *SRE* can be eliminated.

This work is based on the modification of the classical linear estimator *LE*. The core idea of *SLE* is to replace the g' bits in *LE* with g' *DR*. The g' *DR* used by *SLE* is called

Algorithm 4 *SLE_Update*

Input: $SLE(aip)$, bip

1: $i \leftarrow H(bip, g', AL)$
2: $sldr \leftarrow$ point to the $SLDR$ of $SLE(aip)$
3: $DRset(sldr[i])$
4: Return

the sliding linear DR array ($SLDR$). Each DR in $SLDR$ is initialized to $2^z - 1$ at the beginning of the algorithm. Unlike SRE, for each bip in $ST(t, k)$, SLE randomly maps it to one DR in $SLDR$ using the hash function $H(bip, g', AL)$ with random seed parameter AL and sets the value of the DR to zero.

At the end of each time slice, the cardinality of $ST(t, k)$ can be estimated by the weight of SLE. The weight of SLE, denoted by $|SLE|_k$, is the number of DR whose value is less than k in $SLDR$. k is the number of time slices in a sliding time window. $|SLE|_k$ corresponds to the number of '1' bits in LE. SLE estimates the cardinality of $|ST(t, k)|$ based on formula (8).

$$|ST(t,k)| = -g' * ln(\frac{g' - |SLE|_k}{g'}) \tag{8}$$

Using SLE to estimate the cardinality of aip also includes two parts: scanning opposite hosts and cardinality estimation. Let $SLE(aip)$ represent the SLE used by aip. Algorithm 4 describes how to use bip to update the $SLDR$ of $SLE(aip)$.

According to [18], the accuracy of SLE is related to the size of g'. The larger the g' is, the higher the accuracy rate will be. On the contrary, larger g' will also require longer time to compute $|SLE|_k$. Therefore, SLE is only suitable for estimating cardinalities of candidate super points at the end of each time slice.

IV. SUPER POINTS DETECTION AND CARDINALITIES ESTIMATION

An SRE or SLE can estimate the cardinality of a single aip in the sliding time window. However, a one-to-one SRE/SLE allocation would be impractical when the quantity of aip in an $ANet$ is so large that the memory consumption is unacceptable. For this reason, a new SRE and SLE based super point estimation algorithm, sliding rough and linear algorithm ($SRLA$), is provided in this section. Like other algorithms, $SRLA$ uses a fixed amount of memory to detect super points and estimate their cardinalities.

A. SLIDING ESTIMATOR ARRAY AND SUPER POINT DETECTION

Since SRE only uses 8 DR to judge whether a host (with a IPv4 address) is a candidate super point, it is fast and memory efficient. Using SRE and SLE together can estimate the cardinalities of candidate super points more quickly and accurately. A combination of SRE and SLE is called *Sliding Estimator* and is recorded as SE'.

Although a large-scale network contains a huge amount of aip, most of them cannot become super points and hence it is unreasonable and impractical to assign an SE' to each aip. If an SE' can be used by multiple aip simultaneously, a fixed amount of memory can be used to complete the cardinality estimation. For example, we can assign a vector consisting of v' SE', and each aip is randomly mapped to a SE' in this vector. The SE' vector is marked as SEV' and shown in FIGURE 2. For an aip, its cardinality is estimated by its corresponding SE in the vector. This method is called vector estimation method. This method uses a fixed amount of memory and it may lead to overestimating. For example, suppose that aip_1 and aip_2 are mapped to the same SE'. When the sum cardinality of aip_1 and aip_2 is greater than the threshold, both aip_1 and aip_2 will be judged as super points regardless of their individual cardinalities.

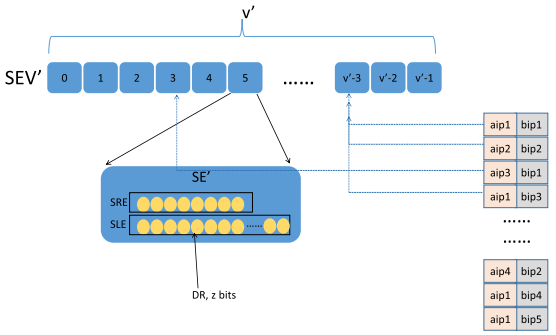


FIGURE 2. Sliding estimator vector.

The estimating error of SEV' is caused by multiple aip using the same SE'. Suppose SEV' has v' SE', the probability that any two aip map to the same SE' is $p_1 = 1/v'$. If SEV' is divided into two vectors SEV'_1 and SEV'_2, the number of SE' of each vector is $v'/2$. But different mapping functions are used, then estimating error would be decreased. In this case, the probability of any two aip mapping to the same SE' is $2/v'$ for both SEV'_1 and SEV'_2, because different mapping functions are used. The probability that they all map to the same SE' in SEV'_1 and SEV'_2 is $p_2 = (\frac{2}{v'})^2 = \frac{4}{v'^2}$, and when $v' > 4$, $p_2 < p_1$. In practice, the value of v' is more than 2^{10}. Obviously, this method can effectively reduce estimating errors.

$SRLA$ is derived by the idea. The SE' vector is transformed into an SE' array of u rows and v columns to estimate the cardinalities of all candidate super points. The SE' array is called *sliding estimator array* (SEA') as shown in FIGURE 3. Each row randomly selects an SE' to record the cardinality of an aip. At the end of a time window, the u SE' corresponding to aip will be merged, and the cardinality of aip will be estimated according to the merged SE'. Since each row in the SEA' uses different mapping functions, the probability that any two aip map to the same SE' in all of these u *rows* is $(\frac{1}{v})^u$.

$SRLA$ is based on sliding estimator array SEA'. When scanning an IP pair $< aip, bip >$, $SRLA$ selects the corresponding SE' in each row by aip, updates the selected SE' with bip. If aip is judged as a candidate super point by SRE, it will be

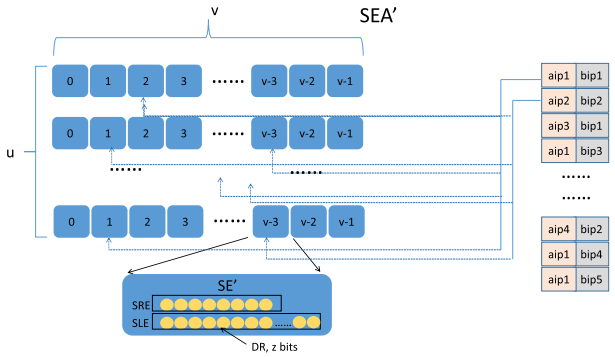


FIGURE 3. Sliding estimator array.

added into a candidate super point list (*CSIP*). When reaching the time window boundary, *SRLA* filters super points from the candidate super point list by estimating their cardinalities using *SLE*. The working process of *SRLA* is as follows:

1). Initialise all data structures;
2). Read IP pair $< aip, bip >$;
3). If not reach the end of a time window, go to step 6);
4). Scan the list of candidate super points *CSIP*, calculate the cardinality of each candidate super point according to *SLE*, filter out the super points;
5). Maintain all *DR* in SEA' by using *DRslide*() (see section III). According to the updated SEA', estimate the cardinality of each candidate super point in *CSIP*, and remove those candidate super points whose cardinality is below the threshold from *CSIP*. The purpose of this step is to generate a list of candidate super points for the next time window.
6). Select one SE from each row of SEA' according to *aip*. This step uses u different hash functions to map *aip* randomly to different SE' in each row.
7). Update the *SLE* and *SRE* in these u SE' found in step 6) by *SRE_Update*() and *SLE_Update*() (see section III) with *bip* as the parameter.
8). If the *SRE* is updated, merge these *SRE* in u SE' to determine whether the *aip* is a candidate super point. If *aip* is a candidate super point, it will be inserted into *CSIP*.
9). Go to step 2).

In step 8), *SRLA* adds these *aip* satisfying the candidate super point condition to *CSIP*. *CSIP* is used to record the *aip* whose cardinality may be higher than the threshold. At the end of the time slice, *SLE* is used to estimate the cardinality of each candidate super point and filter out the super points.

SRLA mainly consists of three parts: firstly, update the core data structure (step 6)-8)). Secondly, estimate cardinality at the end of a time window(step 4)). Finally, maintain the state of the primary data structure and update the CSIP (step 5)) when the window slides. The details are discussed below.

B. UPDATE PRIMARY DATA STRUCTURE

SRLA updates the *SRE* and *SLE* corresponding to *aip* when the IP pair $< aip, bip >$ arrives. When this updating process causes the cardinality of *aip* to exceed the threshold, *aip* will enter *CSIP*. But there may still be multiple *aip*-related IP pairs before the end of the time window. The problem is that *aip* will re-enter *CSIP*, which will lead to excessive *aip* in *CSIP*. Such redundancy is unreasonable and a burden to meet the real-time requirements. To cope with this dilemma, the duplicate candidate super points checking is necessary when adding a candidate super point to *CSIP*.

This operation is done using the principle of the bloom filter. The method is to add a candidate super point indicator, denoted as *SI*, in SE'. *SI* consists of 16 bits. We call the modified structure *SE*, as shown in FIGURE 4.

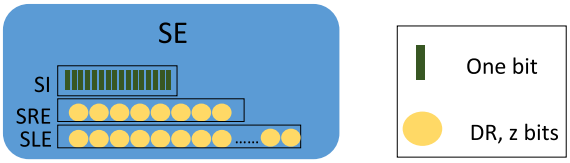


FIGURE 4. Sliding estimator with super points indicator.

A *SE* contains not only one *SRE* and one *SLE*, but also an *SI*. *SI* is used to indicate whether an *aip* has been detected as a candidate super point in a time window. $SI[i]$ denotes the i-th bit in *SI*. When an *aip* is firstly detected as a candidate super point, a bit in *SI* is set to 1. The bit is determined by a random hash function with *aip* as its parameter, which maps *aip* randomly to an integer between 0 and 15. The hash function $H(aip, 16, As)$ with random seed parameter As satisfies the requirements. The last four bits of *aip* can also be used as a hashed value directly since the last four bits of IP address itself has high randomness [36], which can improve the speed of the algorithm. If an *aip* corresponds to a bit of 1 in all its corresponding *SI*, it means that the *aip* is already in *CSIP* and does not need to be re-added. At the end of a time window, when the *CSIP* is updated, the *SI* of candidate super points is updated at the same time. *SRLA* uses the array of *SE* composed of u rows and v columns, recorded as *SEA*. *SRLA* updates *SEA* and *CSIP* when scanning IP pairs, i.e. steps 6) to 8) in section IV-A. The pseudo-code is described in Algorithm 5. In the i-th row, *aip* is mapped to a *SE* by hash function $H(aip, v, AR_i)$ with random seed parameter AR_i. Let $SRE[i, j]$, $SLE[i, j]$ and $SI[i, j]$ represent the *SRE*, *SLE* and *SI* of the *SE* in the i-th row, j-th column of *SEA*.

The purpose of merging *SI* and *SRE* is to eliminate the impact of mapping multiple *aip* to the same *SE* and restore the real *SRE*. Neither the updating operation nor the merging operation involve complex calculation. All IP pairs in a time slice are processed according to the above operations. When the time slice boundary is reached, *CSIP* contains candidate super points in the sliding time window. Using the more accurate estimator *SLE*, we get cardinalities of candidate super points in *CSIP*. How the cardinality is estimated is the problem to be discussed in the next section.

Algorithm 5 ScanIPpair

Input: $< aip, bip >$

for $i \in [0, u-1]$ **do**
  $SLE\_Update(SLE[i, H(aip, v, AR_i)], bip)$
end for
if $SRE\_IsRecord(bip)$ is $FALSE$ **then**
  Return
end if
for $i \in [0, u-1]$ **do**
  $SRE\_Update(SRE[i, H(aip, v, AR_i)], bip)$
end for
$USI \leftarrow$ a new SI with every bit equal to 1
for $i \in [0, u-1]$ **do**
  $USI \leftarrow USI \& SI[i, H(aip, v, AR_i)]$
end for
$siidx \leftarrow$ last 4 bits of aip
if $USI[siidx] == 1$ **then**
  Return
end if
$URE \leftarrow$ a new SRE with every DR equal to 0
for $i \in [0, u-1]$ **do**
  $srdr \leftarrow SRDR$ of $SRE[i, H(aip, v, AR_i)]$
  for $j \in [0, g-1]$ **do**
    $dr \leftarrow srdr[j]$
    $URE[j] \leftarrow DRjoin(URE[j], dr)$
  end for
end for
if $SRE\_IsSP(URE)$ **then**
  $siidx \leftarrow$ last 4 bits of aip
  insert aip into $CSIP$
  for $i \in [0, u-1]$ **do**
    $SI[i, H(aip, v, AR_i)][siidx] \leftarrow 1$
  end for
end if

C. ESTIMATE CARDINALITIES OF CANDIDATE SUPER POINTS

This section discusses how to use SLE to estimate cardinalities of candidate super points in $CSIP$ at the end of a time window. The estimation is based on equation (8). What needs to be solved is how to merge these u SLE corresponding to a candidate super point in the most reasonable way.

After scanning the IP pairs in time slice t, if the value of any DR in SEA is less than k, the DR is set by $DRset()$ operation in time window $W(t-k+1, k)$. At this time, the DR is said to be active in the time window $W(t-k+1, k)$.

To reduce the impact of sharing SE, u SE are utilized together to record and estimate the cardinality of each aip. Using the union SLE, denoted as ULE, to reduce the overestimation of cardinality. For an aip, its ULE, denoted as $ULE(aip)$, is acquired by merging its corresponding u SLE in SEA. However, when a time slice contains sufficiently large number of IP pairs, some DR in the ULE will be active because they are mapped by other aip. Unlike SRE, SLE needs to perform more precise calculations. In the $ULE(aip)$, these false active DR, which become active not because of aip, should be eliminated as much as possible.

Let $|LDR(i)|_k$ denote the number of active DR in all SLE in the i-th row of SEA. Since SEA maps aip randomly to different SLE and bip randomly to DR in a SLE, the distribution of active DR can be considered uniform. In the i-th row, the probability that a DR of SLE is active is $P_{DR}^{SLE}(i) = \frac{|LDR(i)|_k}{g' * v}$. $|LDR(i)|_k$ is obtained by scanning all SLE in the i-th row. Select one SLE from each row in the SEA and merge these u SLE by the $DRjoin()$ operation to get ULE. Let Psu represent the probability that a DR in ULE is active. Psu can be calculated according to the following formula.

$$Psu = \prod_{i=0}^{u-1} P_{DR}^{SLE}(i) \tag{9}$$

Let $|ULE(aip)|'_k$ denote the number of active DR in $ULE(aip)$ which are active due to aip. Let $|ULE(aip)|_k$ denote the number of all active DR in $ULE(aip)$. $|ULE(aip)|_k$ can be obtained by counting the active DR in ULE. But $|ULE(aip)|_k$ contains the number of false active DR. By estimating the number of these false active DR and removing them from $|ULE(aip)|_k$, we can get the estimated value of $|ULE(aip)|'_k$. Obviously, it is more reasonable to estimate the cardinality of aip with $|ULE(aip)|'_k$.

Let $OP(aip, t, k)$ represent the set of opposite host of aip in $W(t, k)$ and $|OP(aip, t, k)|$ represent the number of distinct host in $OP(aip, t, k)$, i.e. the cardinality of aip. According to formula (8), we acquire the following equation.

$$|ULE(aip)|'_k = g' - g' * e^{-\frac{|OP(aip,t,k)|}{g'}} \tag{10}$$

Because in $ULE(aip)$, the remaining $g' - |ULE|'_k$ DR are false active with the probability of Psu, $|ULE(aip)|'_k$ is estimated by the following equation:

$$|ULE(aip)|'_k = |ULE(aip)|_k - (g' - |ULE(aip)|'_k) * Psu \tag{11}$$

By combining formulas (10) and (11), we can get the equation (12) for estimating the cardinality of aip.

$$|OP(aip, t, k)|' = -g' * ln(\frac{g' - |ULE(aip)|_k}{g' * (1 - Psu)}) \tag{12}$$

$SRLA$ uses equation (12) to estimate the cardinalities of candidate super points in $CSIP$. The formula improves the accuracy of cardinality estimation by removing the number of false active DR in ULE. Using this method to remove the noise in ULE can improve the average accuracy of the estimation results.

After estimate the cardinalities of candidate super points, SEA should be updated for the next time window as shown in the next section.

D. UPDATE DATA STRUCTURE AT THE BOUNDARY OF A TIME WINDOW

To run in the sliding time window, *SRLA* must incrementally update *SEA* instead of reinitializing all *SE* every time the window slides forward. After estimating cardinalities, *SRLA* needs to update all *SI*, *DR* and *CSIP* before scanning the IP pairs in the next time slice, that is, step 5) of *SRLA* in the section IV-A.

DR is updated by the *DRslide*() operation in section III. After updating *DR*, the cardinalities of some *aip* in *CSIP* decreases. So *CSIP* is updated based on the updated *SEA* to remove candidate hosts that are no longer determined as candidate super points by *SRE*. And *SI* needs to be reset according to the new *CSIP*. The specific update steps are as follows:

1) Increase all *DR* in *SEA* by 1;
2) Set all *SI* in *SEA* to 0;
3) Update *CSIP* and *SI* by *SRE* in the updated *SEA*: if a candidate super point is no longer judged as a super point by *SRE*, delete it from *CSIP*; else set bits of *SI* which is related with this candidate super point.

SRLA detect super points and estimate their cardinalities as described before. *SRLA* is also a parallel algorithm. In the next section, we shows how to deploy *SLRA* on GPU to deal with 40 Gbps traffic in real-time.

V. EXPERIMENTAL RESULTS

To verify the performance of *SRLA*, this paper experiments on real-world high-speed core network trace collected at the network boundary of Nanjing node of CERNET [34]. The experimental data contain two IP traces, which are collected on October 23, 2017, and March 8, 2018, respectively. They are both one-hour IP traces starting from 13:00 to 14:00 and currently available for download on the IPtas website [34]. The average values of basic information of these two traffics under a 5-minute discrete time window is shown in TABLE 2. In TABLE 2, "*#ANet IP*" and "*#BNet IP*" denote the average number of distinct IP addresses for *ANet* and *BNet* in each window respectively; "*#Flow*" denotes the average number of distinct IP pairs. The threshold value of the super point is 1024, and "*#Super points*" denotes the average number of super points. As can be seen from the table, the super point accounts for only 0.0422% of the total number of hosts in *ANet*.

TABLE 2. Traffic summary.

Traffic	#ANet IP	#BNet IP	#Flow	Packet Speed(kpps)	#Super points
Iptas 2017_10_23	1262184	1588792	15163646	4622.224	598.8333
Iptas 2018_03_08	1406287	1815909	13429067	3141.617	527.4167

The experiment consists of three parts:

1) Test the influence of different combinations of u, v and g';
2) Compare the performance of the *SRLA* with other algorithms under discrete time windows,
3) Run *SRLA* under sliding time windows.

In the experiment part, the standard answer is obtained based on the accurate statistical algorithm. Because *SRLA* can run in parallel conveniently, all the experiments in this paper are carried out on a PC with a GPU(Nvidia Titan XP, 12 GB memory). We describe how to deploy *SRLA* in GPU firstly.

A. DEPLOY ON GPU

When scanning IP pairs, *SRLA* only sets some bits of *SEA* to 1 or sets some *DR* to 0 and the results are the same regardless of the sequence of execution. Therefore, large amount of IP pairs can be processed simultaneously by updating *SEA* and *DR* via multiple threads [37].

GPU is a particular device with rich computing units and high memory throughput [38]. Although CPUs may have slight advantages over GPUs in terms of single computing core [39], the highly concentrated processing units grant GPU significantly more computing power than CPU, especially when processing parallel computing tasks of single instruction stream and multiple data streams [40]. The primary data structure *SEA* used by *SRLA* can be accessed or modified by multiple threads at the same time, and each thread uses the same algorithm to process different IP address pairs. Obviously *SRLA* is suitable for GPU implementation [41]. But since the GPU can only access its dedicated memory [42], IP address pairs need to be stored in the memory pool and then copied to GPU memory.

SEA is allocated on GPU's global memory [43]. Two buffers of the same size are allocated on the server and the GPU separately for storing IP pairs. When the buffer on the server is full, the stored IP pairs are copied to the GPU's buffer through the PCIe bus [44]. The number of IP pairs stored in the buffer determines the number of threads to be initiated in the GPU. After the IP pair buffer replication is completed, the GPU starts a processing thread for each IP pair. Each of these threads runs the steps 6) to 8) of *SRLA* as described in section IV-A.

The set of experimental traffic is saved on the local hard disk. In the experiments, the IP pairs are read from the hard disk file into the server buffer. If the algorithm needs to run in real-time network environment, two or more IP pair buffers can be reserved on the server side to save real-time network traffic, and hence to prevent IP pairs overflow caused by the sudden increase of packets. After processing all IP pairs in a time slice, the GPU adopts another set of threads to calculate the cardinalities of candidate super points in *CSIP* by step 4) in section IV-A. After cardinalities estimation, *SEA* and *CSIP* need to be updated to maintain their states in the next time window. Updating of *SEA* and *CSIP* (step 5) in section IV-A) can also be done in parallel with different threads in the GPU.

In the latter analysis, *ScanT* is used to indicate the time of updating *SEA* according to IP pairs in a time slice, that is, the time of scanning IP pairs. *EstT* represents the time of

cardinality estimation. *SEAT* and *CSIPT* stand for the time of updating *SEA* and *CSIP* after cardinalities estimation at the end of a time window. *SliceT* denotes the length of a time slice. In a time slice, the total running time of *SRLA* is the sum of *ScanT*, *EstT*, *SEAT* and *CSIPT*, which is recorded as *AllT*. *AllT* must be less than *SliceT* to estimate cardinalities of super points in real time. In this experiment, *ScanT*, *EstT*, *SEAT* and *CSIPT* are the running time on GPU. Without special notification, the unit of running time is milliseconds (ms).

B. THE INFLUENCE OF PARAMETERS ON SRLA

The parameter setting of the *SEA* will affect the performance of *SRLA*. This experiment compares the effects of different u, v and g' on *SRLA* and selects the reasonable parameters for the following experiments regarding accuracy, memory occupancy and estimation time. This experiment is done under the discrete time window.

Under the discrete time window, the parameters of *SRLA* are set as follows: the length of each time slice *SliceT* is set to 300 seconds, the number of time slices k is set to 1 in each window, and the number of bits in each *DR* (z) is set to 1. Each traffic is divided into 12 discrete time windows.

Firstly, the accuracy of the algorithm is measured by the false positive rate (*FPR*) and the false negative rate (*FNR*).

Definition 2 (FPR/FNR): For a segment of traffic containing N super points, let N' represent the number of super points detected by an estimation algorithm. In the N' detected super points, there are N^+hosts are not super points, and there are N^- super points not detected out by the estimation algorithm. Then *FPR* is the ratio of N^+ to N; *FNR* is the ratio of N^- to N.

FPR is inversely proportional to *FNR*. Therefore, this paper uses the sum of *FPR* and *FNR*, recorded as the total error rate (*FTR*), to compare the accuracy of an estimation algorithm.

FIGURE 5 and 6 show the average accuracy of SRLA of two traffics under 12 discrete time windows using different parameter combinations.

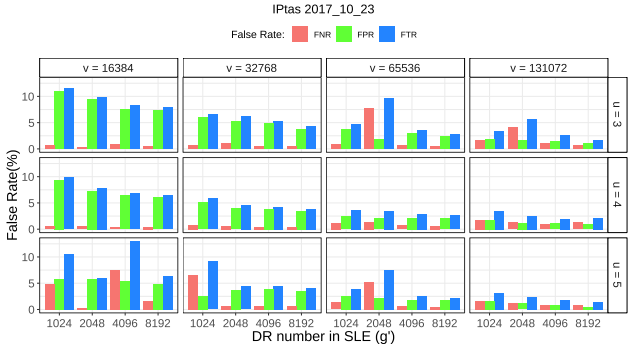


FIGURE 5. Average accuracy of traffic 1.

As can be seen from these two figures, when v is 65536 or 131072 and u is 4 or 5, the *FTR* of *SRLA* is lower than that of u and v with other values. High accuracy is a necessary condition for *SRLA* algorithm to run successfully. To select the parameters that can make *SRLA* obtain high accuracy, we list the combinations of parameters whose average *FTR* is less than 3% in both set of traffic in TABLE 3.

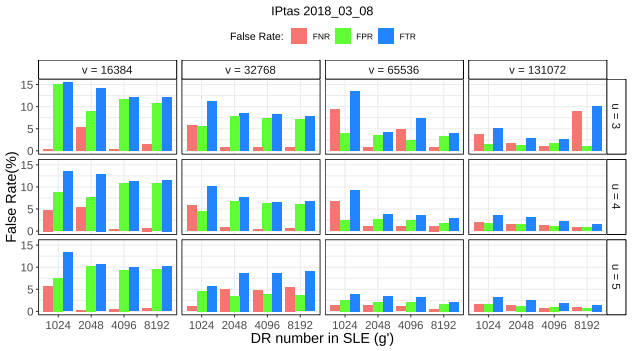


FIGURE 6. Average accuracy of traffic 2.

TABLE 3. Running result of SRLA under different parameters.

u	v	g'	Memory(MB)	FTR(%)	EstT(ms)
4	131072	2048	130	2.7383	1.845
5	131072	2048	162.5	2.523	2.4374
3	131072	4096	193.5	2.4979	2.7384
4	65536	8192	257	2.7616	8.8724
4	131072	4096	258	2.0477	4.6095
5	65536	8192	321.25	2.2227	14.8006
5	131072	4096	322.5	1.7867	4.6624
4	131072	8192	514	1.8362	6.6912
5	131072	8192	642.5	1.4124	8.573

As can be seen from TABLE 3, there are nine combinations of parameter that can make *FTR* less than 3% in both traffics. TABLE 3 also lists the memory occupied by *SRLA* (column "Memory"), the average total error rate (column "FTR") and the average estimation time (column "EstT") of the two traffic, in which "Memory" refers to the memory occupied by the *SEA*.

The data in TABLE 3 are arranged in order of memory occupancy in descending order. Since obviously less memory consumption by *SRLA* is preferable, we make a further analysis of the first three lines, which do not exceed 200 MB of memory. TABLE 4 lists the ratio of *FTR*, *Memory*, and *EstT* to the first row (*FTR* ratio, *Memory* ratio, and *EstT* ratio). As can be seen from this table, *EstT* increases gradually with the increase of memory. Compared with the first line, the memory of the second line increases by 25% while the *FTR* decreases by 7.86%. The memory of the third line increased by 48.85%, but the *FTR* decreased by only 8.78%, only 0.92% more than the second line.

Based on the above analysis of accuracy, *EstT*, and memory occupancy, in the following experiments, v, u, and g' are set to 13172, 5, and 2048 respectively. Under this combination of parameters, the average result of *SRLA* running in 24 discrete time windows is shown in TABLE 5.

TABLE 4. Ratio of different combinations of parameter.

u	v	g'	FTR ratio	Memory ratio	EstT ratio
4	131072	2048	1	1	1
5	131072	2048	0.9214	1.25	1.3211
3	131072	4096	0.9122	1.4885	1.4842

TABLE 5. The performance of SRLA with selected parameters.

	FPR(%)	FNR(%)	FTR(%)	ScanT(ms)	EstT(ms)	AllT(ms)
max	2.4299	2.5	4.1121	2192.03	6.4253	2193.94
min	0.1859	0.1869	1.1945	1373.278	1.7731	1375.053
avg	1.3227	1.2002	2.523	1731.279	2.4374	1733.716

C. COMPARING RESULTS UNDER DISCRETE TIME WINDOWS

To evaluate the performance of *SRLA* under discrete time windows, this paper compares it with double connection degree sketch (*DCDS*) [9], vector bloom filter algorithm (*VBFA*) [10], and grand spread estimator (*GSE*) [11]. TABLE 6 shows the average results of different algorithms under two traffics (there are total 24 discrete time windows). In TABLE 6, "SW" indicates whether the algorithm can run in a sliding time window.

TABLE 6. Performance of different algorithms.

Alg	Memory (MB)	SW	FNR (%)	FPR (%)	FTR (%)	ScanT (ms)	EstT (ms)	AllT (ms)
DCDS	384	No	0	5.278	5.278	2548.128	1442.911	3991.038
VBFA	640	No	0	5.785	5.785	1495.414	462.989	1958.404
GSE	514	No	2.474	1.121	3.595	967.759	988.341	1956.101
SRLA	162.5	Yes	1.323	1.2	2.523	1731.279	2.437	1733.716

As can be seen from TABLE 6, *SRLA* has the highest accuracy and occupies only half of the memory of other algorithms. From the runtime analysis, *SRLA* uses the least total time *AllT*. Moreover, the *EstT* of *SRLA* is only 0.519% to 0.166% of the *EstT* of other algorithms. Since *SRLA* generates a list of candidate super points when scanning IP pairs, which reduces the number of hosts that need to estimate the cardinality, its *EstT* is smaller than other algorithms'. *EstT* plays an essential role under sliding time windows. Fast super point cardinality estimating is a necessary condition for the algorithm to run under sliding time windows because under the sliding time window, cardinality estimation and super point detection will be more frequent. If *EstT* is large, on the other hand, it will affect the efficiency of the algorithm and even make the algorithm unable to deploy in real time. Besides, among these algorithms, *SRLA* is also the only one that can run in sliding time windows.

D. EXPERIMENTS UNDER SLIDING TIME WINDOWS

For the sliding time window experiments, *SliceT* is set to 1 second and k is 300. When k is 300, the number of bits of *DR* (z) could be as small as $ceil(log_2(300)) = 9$ bits. To make the operation of *DR* simple, in this experiment, *DR* occupies continuous bytes, that is, z is set to 16. In this experiment, the error rate and running time of *SRLA* are tested under sliding time windows. In each traffic, the window slides from $W(0, 300)$ to $W(3300, 300)$, i.e. each traffic contains 3301 time windows and each time window is 5 minutes.

TABLE 7 lists the maximum, minimum, average, and variance of error rate (FNR, FPR, FTR) in two traffic (each traffic has 3301 time windows) under sliding time windows. "All traffics" denotes the union of all time windows in the two traffic, with a total of 6602 time windows. Under the sliding time window, since each traffic contains a large number of time windows, we list the variance of SRLA results, which can be used to observe the fluctuation of results in different time windows.

TABLE 7. False rate under sliding time windows.

	IPtas 2017_10_23			IPtas 2018_03_08			All traffics		
	FPR(%)	FNR(%)	FTR(%)	FPR(%)	FNR(%)	FTR(%)	FPR(%)	FNR(%)	FTR(%)
max	3.2149	2.4834	4.7377	2.4762	2.8571	4.7348	3.2149	2.8571	4.7377
min	0.1715	0	0.6557	0.1898	0	0.5929	0.1715	0	0.5929
avg	1.3451	1.0011	2.3462	1.1698	1.1069	2.2767	1.2574	1.054	2.3114
var	0.2267	0.1757	0.3722	0.1493	0.1844	0.3407	0.1956	0.1828	0.3576

Under sliding time windows, the average *FTR* of *SRLA* is 2.3114%, which is close to the error rate of discrete time windows (2.523%). It shows that *SRLA* can obtain a higher accuracy under sliding time windows than that under discrete time windows. *SRLA* also needs to update *SEA* and *CSIP* at the end of each time window, compared with running under discrete time windows. Consequently, under the sliding time window, the running time of *SRLA* also includes *SEAT* and *CSIPT*. TABLE 8 lists the maximum, minimum, average, and variance of running time (*ScanT*, *EstT*, *SEAT*, *CSIPT*) in 6602 time windows (the union of time windows in two traffics) under sliding time windows.

ScanT in the sliding time window is smaller than that in discrete time windows. The reason is that *SRLA* can update *SEA* incrementally under sliding time windows. Each time window only needs to scan IP pairs in one time slice (1 second), and the time slice length under the sliding time window is smaller than the time slice length under discrete time windows (300 seconds). The maximum *AllT* used by *SRLA* in each time slice is only 109.3656 milliseconds, which is less than the length of one time slice (1 second). Hence *SRLA* can run continuously on GPU in real time. And *SRLA* runs smoothly in each time window. The variances in tables 7 and 8 represent the fluctuations of *SRLA* over different time windows. The smaller the variance is, the lower the fluctuation of the results of the algorithm is, and the more stable the *SRLA* stays. In the two traffics (6602 time windows in total), the variance of *FTR* of *SRLA* is only 0.3576. It shows that *SRLA* detects the super points with an error rate closing

to 2.3114%. The variance of *AllT* is only 4.295. The variance of *CSIPT* is even less than 10^{-5} (the values in TABLE 8 arrive accurately decimally hind 4, hence variance of *CSIPT* is 0 in TABLE 8). The experiments show that *SRLA* can output accurate results with a stable running time under sliding time windows.

TABLE 8. Running time under sliding time windows.

	ScanT(ms)	EstT(ms)	SEAT(ms)	CSIPT(ms)	AllT(ms)
max	6.2474	10.5431	92.8992	0.0901	109.3656
min	3.6759	8.7893	90.3204	0.0191	103.0698
avg	4.7506	9.4253	91.6465	0.0207	105.8431
var	0.6848	0.4335	0.5391	0	4.295

VI. CONCLUSION

Real-time acquisition of super point information is a valuable task in the field of network management and network security, hence the related research work has been continuing. Incremental updating and low estimation time are two difficulties. The *SRLA* algorithm proposed in this paper solves this problem for the first time. *SRLA* uses a new data structure *DR* to incrementally record the cardinality. Its structural characteristics enable it to be implemented under the sliding time window conditions. Another feature of *SRLA* is that it designs a lightweight cardinality estimator *SRE*. *SRE* takes up less memory and has fast processing speed. Hence it can detect candidate super points under the condition of satisfying the real-time requirement when scanning IP pairs. When reaching the time window boundary, *SLE*, which has higher accuracy, is used to estimate the cardinality of each candidate super point. *SRLA* is also a parallel algorithm. When running on GPU, *SRLA* can detect the super points in a 40 Gbps high-speed network in real time under sliding time windows. In the further work, we will analyze the super point found by *SRLA* and study its application in network security and management.

REFERENCES

[1] H. Guo, Z. Yang, L. Zhang, J. Zhu, and Y. Zou, "Joint cooperative beamforming and jamming for physical-layer security of decode-and-forward relay networks," *IEEE Access*, vol. 5, pp. 19620–19630, 2017.

[2] S. Chen, Y. Qiao, S. Chen, and J. Li, "Estimating the cardinality of a mobile peer-to-peer network," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 9, pp. 359–368, Sep. 2013.

[3] D. Yin, L. Zhang, and K. Yang, "A DDoS attack detection and mitigation with software-defined Internet of things framework," *IEEE Access*, vol. 6, pp. 24694–24705, 2018.

[4] W. Liu, W. Qu, G. Jian, and L. Keqiu, "A novel data streaming method detecting superpoints," in *Proc. IEEE INFOCOM WKSHPS*, Apr. 2011, pp. 1042–1047.

[5] A. T. Liem, I. Hwang, A. Nikoukar, C.-Z. Yang, M. S. Ab-Rahman, and C. Lu, "P2P live-streaming application-aware architecture for QoS enhancement in the EPON," *IEEE Syst. J.*, vol. 12, no. 1, pp. 648–658, Mar. 2018.

[6] Y. Cao, Y. Gao, R. Tan, Q. Han, and Z. Liu, "Understanding internet DDoS mitigation from academic and industrial perspectives," *IEEE Access*, vol. 6, pp. 66641–66648, 2018.

[7] H. Peng, Z. Sun, X. Zhao, S. Tan, and Z. Sun, "A detection method for anomaly flow in software defined network," *IEEE Access*, vol. 6, pp. 27809–27817, 2018.

[8] K. Sood, S. Yu, Y. Xiang, and H. Cheng, "A general QoS aware flow-balancing and resource management scheme in distributed software-defined networks," *IEEE Access*, vol. 4, pp. 7176–7185, 2016.

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

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

[11] S.-H. Shin, E.-J. Im, and M. Yoon, "A grand spread estimator using a graphics processing unit," *J. Parallel Distrib. Comput.*, vol. 74, no. 2, pp. 2039–2047, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731513002189

[12] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM J. Comput.*, vol. 31, no. 6, pp. 1794–1813, 2002.

[13] W. Lin, X. Xiao, X. Xie, and X. Li, "Network motif discovery: A GPU approach," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 3, pp. 513–528, Mar. 2017.

[14] J. Xu, W. Ding, J. Gong, X. Hu, and S. Sun, "SRLA: A real time sliding time window super point cardinality estimation algorithm for high speed network based on GPU," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.*, Jun. 2018, pp. 942–947.

[15] S. Mori, A. Sato, and K. Yoshida, "Enhancing performance of cardinality analysis by packet filtering," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2016, pp. 23–28.

[16] C. T. Nguyen, T. T. Hoang, and V. X. Phan, "A simple method for anonymous tag cardinality estimation in RFID systems with false detection," in *Proc. 4th NAFOSTED Conf. Inf. Comput. Sci.*, Nov. 2017, pp. 101–104.

[17] Q. Xiao *et al.*, "Cardinality estimation for elephant flows: A compact solution based on virtual register sharing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3738–3752, Dec. 2017.

[18] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[19] L. Huang, Q. Yang, and W. Zheng, "Online hashing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2309–2322, Jun. 2018.

[20] Y. Liu, W. Chen, and Y. Guan, "Identifying high-cardinality hosts from network-wide traffic measurements," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 5, pp. 547–558, Sep./Oct. 2016.

[21] B. Silva and G. Fraidenraich, "Performance analysis of the classic and robust chinese remainder theorems in pulsed Doppler radars," *IEEE Trans. Signal Process.*, vol. 66, no. 18, pp. 4898–4903, Sep. 2018.

[22] H. Chen, J. Sun, L. He, K. Li, and H. Tan, "BAG: Managing GPU as buffer cache in operating systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1393–1402, Jun. 2014.

[23] X. Zhou, W. Liu, Z. Li, and W. Gao, "A continuous virtual vector-based algorithm for measuring cardinality distribution," in *Proc. IAAA*, Aug. 2014. pp. 43–53.

[24] E. Alasadi and H. Al-Raweshidy, "OLC: Open-level control plane architecture for providing better scalability in an SDN network," *IEEE Access*, vol. 6, pp. 34567–34581, 2018.

[25] J. Erman and K. Ramakrishnan, "Understanding the super-sized traffic of the super bowl," in *Proc. Conf. Internet Meas. Conf.* New York, NY, USA: ACM, Aug. 2013, pp. 353–360. doi: 10.1145/2504730.2504770.

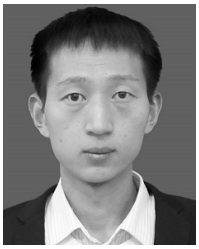
[26] W. Liu, C. Liu, and S. Guo, "A hash-based algorithm for measuring cardinality distribution in network traffic," *Int. J. Auton. Adapt. Commun. Syst.*, vol. 9, nos. 1–2, pp. 136–148, Mar. 2016. doi: 10.1504/IJAACS.2016.075387.

[27] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proc. 19th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.* New York, NY, USA: ACM, Aug. 2010, pp. 41–52. doi: 10.1145/1807085.1807094.

[28] M. G. Khoshkholgh, V. C. M. Leung, and K. G. Shin, "Fast and accurate cardinality estimation in cellular-based wireless communications," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2015, pp. 1119–1123.

[29] B. Li, Y. He, and W. Liu, "Towards constant-time cardinality estimation for large-scale RFID systems," in *Proc. 44th Int. Conf. Parallel Process.*, Sep. 2015, pp. 809–818.

[30] Y. Zheng, X. Wang, D. Yang, and S. Ding, "An efficient RFID tag cardinality estimation protocol based on bit detection," in *Proc. IEEE 17th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2017, pp. 602–606.
[31] J. Xu, W. Ding, J. Gong, X. Hu, and J. Liu, "High speed network super points detection based on sliding time window by GPU," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. with Appl.*, Dec. 2017, pp. 566–573.
[32] J. Shan, J. Luo, G. Ni, Z. Wu, and W. Duan, "CVS: Fast cardinality estimation for large-scale data streams over sliding windows," *Neurocomputing*, vol. 194, pp. 107–116, Jun. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231216002320
[33] J. Shan, Y. Fu, G. Ni, J. Luo, and Z. Wu, "Fast counting the cardinality of flows for big traffic over sliding windows," *Frontiers Comput. Sci.*, vol. 11, no. 1, pp. 119–129, Feb. 2017. doi: 10.1007/s11704-016-6053-x.
[34] CERNET. (2017). *China Education and Research Network*. [Online]. Available: http://iptas.edu.cn/src/system.php
[35] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Per-flow counting for big network data stream over sliding windows," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10.
[36] L. Zhang, Q. Deng, Y. Su, and Y. Hu, "A box-covering-based routing algorithm for large-scale SDNs," *IEEE Access*, vol. 5, pp. 4048–4056, 2017.
[37] Y. Suzuki, Y. Fujii, T. Azumi, N. Nishio, and S. Kato, "Real-time GPU resource management with loadable kernel modules," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1715–1727, Jun. 2017.
[38] J. Mielikainen, E. Price, B. Huang, H. A. Huang, and T. Lee, "Gpu compute unified device architecture (cuda)-based parallelization of the rrtmg shortwave rapid radiative transfer model," *IEEE J. Sel. Topics Appl. Earth Observat. Remote Sens.*, vol. 9, no. 2, pp. 921–931, Feb. 2016.
[39] A. Vilches, A. Navarro, R. Asenjo, F. Corbera, R. Gran, and M. J. Garzarán, "Mapping streaming applications on commodity multi-CPU and GPU on-chip processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1099–1115, Apr. 2016.
[40] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Communication and load balancing optimization for finite element electromagnetic simulations using multi-GPU workstation," *IEEE Trans. Microw. Theory Techn.*, vol. 65, no. 8, pp. 2661–2671, Aug. 2017.
[41] D. Foley and J. Danskin, "Ultra-performance pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, Mar./Apr. 2017.
[42] G. Chen, X. Shen, B. Wu, and D. Li, "Optimizing data placement on GPU memory: A portable approach," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 473–487, Mar. 2017.
[43] S. Mittal, "A survey of techniques for architecting and managing GPU register file," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 16–28, Jan. 2017.
[44] G. Chen, B. Wu, D. Li, and X. Shen, "Enabling portable optimizations of data placement on GPU," *IEEE Micro*, vol. 35, no. 4, pp. 16–24, Jul. 2015.



JIE XU received the B.S. degree in international trade and economy from Jiangsu Normal University, Jiangsu, China, in 2011, and the M.S. degree in computer science and technology from Yangzhou University, Jiangsu, China, in 2014. He is currently pursuing the Ph.D. degree in computer science and technology with Southeast University, Jiangsu. His research interests include data mining, supercomputing, distributed computing, and networking security.

WEI DING received the B.S. degree in computer science from Nanjing University, Nanjing, China, in 1982, and the M.S. degree in system engineering and the Ph.D. degree in computer science and technology from Southeast University, Nanjing, in 1987 and 1995, respectively, where she is currently a Professor with the School of Cyber Science and Engineering. Her research interests include network management and network security.

QIUSHI GONG received the B.S. degree in electrical engineering from Southeast University, Nanjing, China, in 2010, the M.S. degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 2013, and the Ph.D. degree in communication engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2018. He is currently a Lecturer with the School of Cyber Science and Engineering, Southeast University. He is working on the transmission and security of fine-grained scalable video coding.

XIAOYAN HU (M'18) received the Ph.D. degree in computer architecture from Southeast University, Nanjing, China, in 2015. She visited the NetSec Laboratory, Colorado State University, a research group working on NDN, from 2010 to 2012. She is currently an Assistant Professor with the School of Cyber Science and Engineering and the School of Computer Science and Engineering, Southeast University. Her research interests include future network architecture and network security.

HAIQING YU received the bachelor's degree in management from the School of Economics and Management, Changchun University of Science and Technology, Changchun, Jilin, in 2014. He is currently pursuing a master's degree in computer technology with the School of Cyber Security, Southeast University, Nanjing, China. His research interests include network measurement and network management, and intelligent detection of network attacks and defenses.