Hindawi
Security and Communication Networks
Volume 2022, Article ID 1258482, 14 pages
https://doi.org/10.1155/2022/1258482

Research Article

Network Host Cardinality Estimation Based on Artificial Neural Network

Xu Jie,[1] **Lan Haoliang**,[1] **Ding Wei**,[2] **and Ju Ao**[1]

[1]*Jiangsu Police Institute, Nanjing 210031, China*
[2]*Southeast University, Nanjing 211102, China*

Correspondence should be addressed to Xu Jie; xujieip@163.com

Received 30 October 2021; Revised 12 February 2022; Accepted 19 February 2022; Published 24 March 2022

Academic Editor: Junggab Son

Cardinality estimation plays an important role in network security. It is widely used in host cardinality calculation of high-speed network. However, the cardinality estimation algorithm itself is easy to be disturbed by random factors and produces estimation errors. How to eliminate the influence of these random factors is the key to further improving the accuracy of estimation. To solve the above problems, this paper proposes an algorithm that uses artificial neural network to predict the estimation bias and adjust the cardinality estimation value according to the prediction results. Based on the existing algorithms, the novel algorithm reduces the interference of random factors on the estimation results and improves the accuracy by adding the steps of cardinality estimation sampling, artificial neural network training, and error prediction. The experimental results show that, using the cardinality estimation algorithm proposed in this paper, the average absolute deviation of cardinality estimation can be reduced by more than 20%.

1. Introduction

From the early single LAN to today's Internet, the development speed of the network is faster and faster, and the network traffic is also larger and larger [1]. How to calculate the relevant network attributes in real time from the massive network data is the key issue concerned by network managers and network security researchers. Host cardinality [2] is one of the most important network attributes. Specifically, the host cardinality refers to the cardinality of an IP address in the network, that is, the number of hosts communicating with the IP address in a period of time.

Host cardinality is an important network security attribute, and many network security problems are related to it. For example, in a DDoS attack, the victim server will receive a large number of attack packets from different hosts. At this time, the cardinality of the victim server will suddenly increase. From the change of host cardinality, we can see whether a DDoS attack is on. Another case is network scanning. Hackers will detect the network and scan other hosts in the network before launching an attack. At this time, the scanner will initiate connections to a large number of different hosts, which will also lead to a sudden increase in the cardinality of the scanner. Through the real-time monitoring of the host cardinality, the above attacks can be found in time.

The host cardinality is not a simple count of how many IP messages the host receives or sends in a period of time. An IP address can send multiple packets to or receive multiple packets from the same host within a period of time. Therefore, when calculating the host cardinality, it is necessary to remove the duplicate IP addresses.

The early cardinality calculation method saved the IP address of each host and the IP address communicated with it in memory by some famous data structure such as binary tree [3]. When an IP packet appears, first look for the IP address in the packet. If the IP address is not in the memory, add it to the memory. If the IP address is already in the memory, directly scan the next IP packet. In order to improve the retrieval speed, data structures such as red and black trees [4] can be used to save IP addresses. After scanning all IP messages in a time window, the cardinality of

each host can be obtained by counting the number of IP addresses in memory. The advantage of this method is that the calculation is accurate, and there is no error. Snort [5], a famous intrusion detection system, uses this method to calculate the host cardinality. However, since each IP address needs to be saved, the memory footprint increases with the increase of different IP addresses. And the time to retrieve each IP address in memory will also increase with the increase of the number of IP addresses. For hosts with cardinality m, the time complexity is at least $O(\log_2(n) * \log_2(m))$, and n is the number of all hosts. Therefore, the traditional cardinality calculation method is only suitable for small-scale networks or offline calculation of the accurate value of cardinality to compare the accuracy of other algorithms, but not for real-time cardinality calculation of high-speed networks.

On high-speed networks, such as edge of country-wide network [6], each IP address cannot be saved and retrieved in real time; hence, the host cardinality is usually calculated by estimation. When calculating the host cardinality, the estimation algorithm uses a fixed amount of memory to process each packet with $0(1)$ time complexity. Compared with the traditional cardinality calculation method, the cardinality estimation algorithm occupies less memory and has a fast calculation speed. It is suitable for real-time processing of high-speed network data. However, the results of the cardinality estimation algorithm will have errors. How to reduce the error of estimation results is one of the research topics.

In the cardinality estimation algorithm, the factors affecting the accuracy of the estimation include the distribution characteristics of network flow and the parameters of random functions [7] used by the algorithm. If the impact of these random factors on the estimation results can be predicted, the estimation results can be adjusted by removing the predicted estimation error so as to reduce the impact of random factors and improve the estimation accuracy. Inspired by the above ideas, this paper takes the data used in cardinality estimation as the attributes used in cardinality correction (called revision attributes), uses an artificial neural network algorithm (ANN) [8] to learn the relationship between revision attributes and estimating cardinality, and uses the learned model to revise cardinality estimation result.

Because it can automatically learn and process high-dimensional complex data, ANN is applied in many fields and scenes [9], such as speech and image recognition, language translation, and automatic driving. ANN uses the label data to learn the model parameters so as to predict the unknown data. The error between the estimated cardinality and the actual value is called estimation error. The estimation error is affected by the estimation algorithm and network traffic, and it is a random variable. Therefore, we can use ANN to predict this error and then improve the accuracy of the estimation results. This paper studies how to use ANN to realize high-precision cardinality estimation and gives the specific algorithm flow and experimental results. The main contributions of this paper are as follows:

(i) Proposed a novel method of generating cardinality training data set

(ii) Proposed a new algorithm using an artificial neural network to improve the accuracy of cardinality estimation

(iii) Verified the effectiveness of ANN in improving the accuracy of cardinality estimation by experiments

This paper is organized as follows. In Section 2, the existing works are introduced. In Section 3, the way to generate training data for cardinality estimation revision is introduced. In Section 4, we describe how to improve the estimation accuracy by neural network. Section 5 gives the results and analysis of experiments on real-world traffic, and Section 6 summarizes this paper.

2. Related Works

Cardinality calculation is widely used in many domains, such as database [10] and sensor network [11]. Unlike the application in database and sensor network, the cardinality estimation problem in the network is not to estimate a single host's cardinality but to estimate each host's cardinality at the same time [12]. Figure 1 describes the model of network cardinality estimation.

Suppose there are two networks: network A (denoted as ANet) and network B (denoted as BNet), which communicate through router R. Let a1, a2, ... represent the host in ANet and b1, b2, ... represent hosts in BNet.

Definition 1. Peer host: for a host a1 in ANet, the host in BNet communicating with a1 through R (sending data packets to or receiving data packets from a1) in a time window is called the peer host of a1.

Definition 2. Host cardinality: for a host a1 in network ANet, the number of peer hosts of a1 in a time window is called the cardinality of a1 in that time window.

An IP address pair in the form of < **aip, bip** > can be extracted from each packet passing through R, as shown in Figure 1. Each time window will contain several IP address pairs, from which the cardinality of different hosts in the time window can be calculated. For example, if 10 IP packets are received or sent by the host **aip**1 in a time window and the hosts in BNet of these packets are {**bip1, bip2, bip1, bip3, bip6, bip2, bip1, bip9, bip1, bip7**}, the cardinality of aip1 in the time window is 6 (there are 6 different hosts {**bip1, bip2, bip3, bip6, bip7, bip9**} communicating with aip1 in the time window). For the convenience of description, this paper only focuses on the calculation of cardinalities of hosts in network A and assumes that peer hosts of each host in network A only include hosts in network B.

There are two methods for cardinality calculation: accurate statistics method and estimation method. The statistical method saves each peer host of each network A host, and the host cardinality can be accurately calculated at the end of the time window. However, this method needs to save

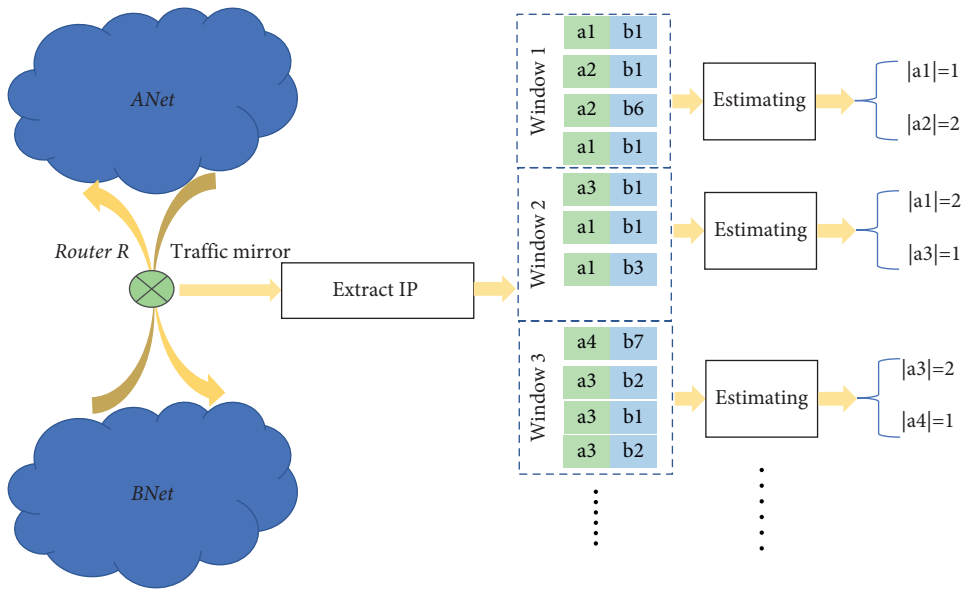

Figure 1: Calculate network host cardinalities from the edge router.

the peer host, which will occupy too much memory and computing time for large-scale networks.

The estimation method uses a fixed-length data structure to estimate the host cardinality. Compared with the statistical method, the estimation method occupies less memory and is more speed [13], but the calculation results will have errors. The statistical counter algorithm is suitable for low-speed small-scale networks or offline computing to obtain the baseline for the comparison of other algorithms. The estimation rule is suitable for real-time computing the host cardinality in high-speed and large-scale networks[14]. Let **RC(ip1)** and **EC(ip1)** represent the real cardinality and estimated cardinality of the host whose IP address is **ip1**, respectively.

The idea of cardinality estimation algorithm is to estimate the cardinality of multiple hosts by using a small amount or even a fixed size of memory [15]. How to estimate the cardinality of a single host is the basis of multihost cardinality estimation [16]. The algorithm used to estimate the cardinality of a single host is called an estimator in this paper. Many estimators have been proposed, such as PCSA [17], loglog/hyperloglog [14,18], and linear estimator (LE) [19]. LE is widely used because of its high accuracy and simple calculation.

2.1. Linear Estimator. LE uses a fixed number of bits to estimate the cardinality. Let g represent the number of bits in a LE. At the beginning of the time window, all bits are reset; that is, the value of every bit is 0. The calculation concept of LE is to map each IP address (the IP address communicating with a host) to a bit and set the bit; that is, the bit value becomes 1. The same IP is mapped to the same bit, which ensures that duplicate IP addresses are recorded only once. The more the bits in the bit string with a value of 1 are, the more the IP addresses appear. At the end of the time window, LE estimates the host cardinality according to the number of bits with value 1. The specific calculation process of LE includes the following steps:

Step 1: at the beginning of the time window, initialize an array composed of **g** bits and set the value of each bit to 0.

Step 2: for each peer host, select a bit in the array and set the bit to 1.

Step 3: at the end of the time window, calculate the number of 0 bits in the array and estimate the host cardinality according to formula (1), where n_0 is the number of 0 bits in the bit vector.

$$\text{Est} = -g * \log\left(\frac{n_0}{g}\right). \quad (1)$$

It can be seen from the calculation process of LE that, for each IP address, LE can be processed in a constant time (Step 2); that is, the time complexity of LE processing each packet is O (1). Compared with the cardinality calculation method based on statistics, the time for LE to process each IP address is independent of the number of different IP addresses and will not increase with the growth of the number of different IP addresses.

3. Several Cardinalities Estimation

Linear estimator can only be used to estimate the cardinality of a host, but there are lots of IP addresses in the network to estimate the cardinality. If a linear estimator is allocated to each IP address, it will waste a lot of storage space, and the calculation process is complex. LE array algorithm (LAA) uses a fixed number of linear estimators to estimate all hosts cardinalities at the same time, and the time complexity of processing each packet is still **O(1)**. Many algorithms are based on LAA, such as Double Connection Degree Sketch

algorithm (DCDS) [20], Vector Bloom Filter Algorithm (VBFA) [21], and Compact Spread Estimator (CSE) [22]. LAA algorithm maps each host to multiple LEs, and each LE in LAA is used to store cardinality information of multiple hosts. The process of the LAA algorithm is listed in the following:

Step 1: at the beginning of the time window, initialize the LE array, which contains α rows and β columns.

Step 2: map each host in network A to an LE in each row of LEA and use these α LEs to record the occurrence of their peer hosts.

Step 3: at the end of the time window, calculate the cardinality of the host **aip** in network A as follows:

Step 3.1: find out the α LEs in the LEA used to record the occurrence of **aip**'s peer hosts.
Step 3.2: combine the α LEs by "bit-AND" operation to obtain a new LE, which is recorded as **ULE**.
Step 3.3: calculate the number of 0 bits in **ULE**, recorded as n_0, and estimate the cardinality of **aip** according to formula (1).

LAA algorithm enables each LE to be shared by multiple hosts in network A, thus reducing the memory occupation. Each host in network A has several LEs to estimate its cardinality. The use of several LEs reduces the error caused by LE sharing.

However, the estimation error of LAA will still be affected by some random factors, such as the random function used by LE itself, the function of mapping each host to LEs in LE array, and the distribution of hosts in network B. Combining multiple LEs does not reduce the impact of these random factors. Artificial neural network is good at learning the relationship between unknown variables and removing cardinality estimation error [10,23,24].

This paper will predict and reduce the error of estimation results through the deep learning method so as to improve the estimation accuracy.

4. Improving Estimation Accuracy by Artificial Neural Network

Deep learning can automatically predict the impact of the hidden factors. The error caused by random factors in the LAA algorithm could be learned by the deep learning method, and the prediction results are adjusted to obtain higher estimation accuracy. Based on this principle, this paper proposes a cardinality estimation algorithm based on an artificial neural network, Neural Network Cardinality Estimation (N2CE). N2CE includes the following 6 steps:

Step 1: scan IP pairs

Step 2: estimate each host's cardinality

Step 3: generate sampling IP sequence

Step 4: construct training data set

Step 5: train neural network

Step 6: predict the estimation error and adjust the estimation result

Step 1 and Step 2 are the same as the existing cardinality estimation algorithms. Step 3 to Step 6 are novel in the N2CE algorithm, which are used to reduce estimation error. Figure 2 illustrates the relationship between each step of N2CE.

First, scan the IP packets in a time window in Step 1, update the LE array according to the IP pair of each IP packet, and record the appearing of the peer host. At the end of the time window, estimate the cardinality of all hosts according to the LE array and the IP address list, and save the estimation results. After estimating the hosts' cardinalities, N2CE predicts the estimation error and adjusts the estimation result in Steps 3 to 6. Steps 3 through 6 will be described in detail in this section.

4.1. Generating Sampling IP Sequence. To predict the host cardinality using the deep learning algorithm, we first need a training data set for learning. Specifically, it requires a data set composed of estimating cardinality and accurate cardinality. The estimating cardinality is used as the attribute of training data, and the bias between accurate cardinality and estimating cardinality is used as the training goal. For each host in network A, the exact cardinality of each host in network A cannot be known because no accurate algorithm is used to save all peer hosts when scanning IP address pairs. Therefore, the host cardinality estimation in network A cannot be directly used as the training data set.

In order to obtain the training data set, this paper uses the sampling method to construct some hosts with definite cardinality. These hosts with certain cardinalities are not hosts in network A, but randomly generated hosts that are not in network A. These hosts are called sampling hosts. The peer hosts of each sampling host are different IP addresses randomly selected from network B. The sampling IP sequence is the collection of these sampling hosts, which is recorded as SIP. Given the cardinality and peer IP, the estimated value, estimation error, and even combined LE of the sampling host can be obtained. Therefore, the training data set can be generated by the sampling host. Before generating training data, it is necessary to determine the number of sampling hosts in the sampling sequence and the cardinality of each sampling host.

The cardinality of different hosts in network A may range from 1 to tens of thousands. If each cardinality is sampled, a lot of resources will be wasted. Moreover, if the cardinality distribution deviation is too large, it will cause overfitting in the neural network training process and reduce the accuracy of the results. Therefore, this paper proposes a method to determine the sampling sequence according to the estimated cardinalities of hosts in network A.

Firstly, hosts in network A are grouped according to their estimating cardinalities. Each group is called an estimating cardinality group. Each cardinality estimation group generates a sampling IP sequence, and each sampling IP sequence is trained to generate its own neural network. Finally, the neural network of the packet is used to predict the cardinality estimation error of the host in the group. Let ECG represent the set of all cardinality estimate groups, ECG_i represents the ith

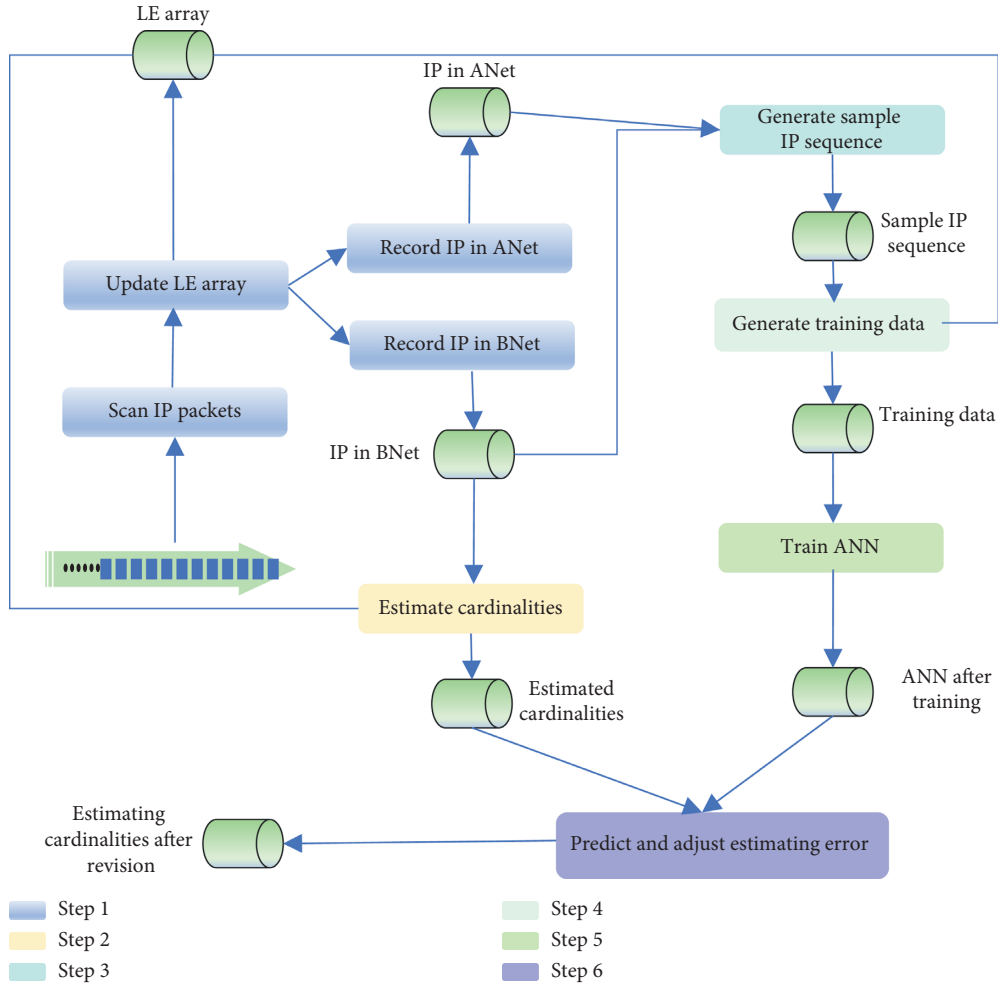

FIGURE 2: Adjustment model of host cardinality estimation based on artificial neural network.

cardinality estimate group. The lower bound and upper bound of ECG_i are defined as follows.

Definition 3. Lower bound: for a cardinality estimate group, the minimum cardinality in it is its lower bound.

Definition 4. Upper bound: for a cardinality estimate group, the maximum cardinality in it is its upper bound.

Let $\mathbf{bt_i}$ represent the cardinality lower bound of the ith group and $\mathbf{tp_i}$ represent the cardinality upper bound of the ith group. ECG_i consists of three parts, $\mathbf{bt_i}$, $\mathbf{tp_i}$, and the set of hosts in network A whose estimating cardinalities are between $[\mathbf{bt_i}, \mathbf{tp_i}]$.

Without omitting any hosts in network A, the estimating cardinality of any host must fall within an ECG. At the same time, in order to ensure that the estimation error of a host is not repeatedly predicted, any host may only belong to one ECG. Therefore, the estimating cardinality range of different ECG is nonintersect, that is, $\mathbf{tp_i} < \mathbf{bt_{i+1}}$. The number of different cardinality values contained in an ECG is called the length of the ECG and is recorded as EGL. Let EGL_i represent the length of the ith ECG; then $EGL_i = \mathbf{tp_i} - \mathbf{bt_i} + 1$. $\mathbf{bt_i}$ and $\mathbf{tp_i}$ can be determined according to the estimating cardinality distribution so that the estimating cardinality of the host in the same group is normally distributed. However, the calculation of this method is complex and needs a lot of extra time. This paper presents a general grouping calculation method with fixed EGL. Its calculation process is shown in Algorithm 1.

Algorithm 1 first sorts the hosts in network A according to their estimating cardinalities from small to large to find the min and max estimation value (record as MinEC and MaxEC). Then, starting from MinEC, hosts are added to different cardinality estimation groups. The lower bound of the first cardinality estimation group is MinEC. Since the EGL is fixed here, the upper bound of the first cardinality estimate group is MinEC + EGL-1. When the estimating cardinality of a host **aip** is found to be greater than the upper bound of the current packet, it indicates that **aip** belongs to the next cardinality estimation group. Then, set **EC(aip)** as the lower bound of the next cardinality estimation group and **EC(aip)**+EGL-1 as the upper bound of the next cardinality estimation group, and take the next cardinality estimation group as the current group. According to Algorithm 1, all hosts in AIP can be divided into different cardinality estimation groups.

(i) **Input** : hosts set in ANet : AIP = {**aip$_1$,aip2,**. . .};
(ii) The length of the group: EGL
(iii) **Output** : ECG//group of hosts in ANet
(1) n ← the number of hosts in AIP
(2) AIP' = {**aip'$_1$,aip'$_2$,**. . .**aip'$_n$**} ← Sort the AIP from small to large according to their estimating cardinality
(3) minEC ← The minimum estimating cardinality of hosts in ANet
(4) j ← 1//*j* is the index of group
(5) bt ← minEC//bt is the lower bound of current group
(6) tp ← bt + EGL-1//tp is the upper bound of current group
(7) For i = 1 to n
(8) If(**EC**(**aip'$_i$**)≤ tp):
(9) Insert **aip'$_i$** into ECG$_i$
(10) Else
(11) j = *j*+1
(12) bt←**EC**(**aip'$_i$**)
(13) tp ← bt + EGL-1
(14) Insert **aip'$_i$** into ECG$_i$
(15) Return ECG

Algorithm 1: Group hosts in ANet.

For example, suppose the interval length EGL is 4, there are 10 hosts in the AIP, and their cardinality estimates are {3, 4, 3, 5, 9, 10, 13, 9, 10, 15}. According to algorithm 1, these cardinality estimates are sorted from small to large, and the sorted estimating cardinalities are {3, 3, 4, 5, 9, 9, 10, 10, 13, 15}. Because the minimum estimating cardinality is 3, the lower bound of the first cardinality estimation group is 3, and the upper bound is 6. Therefore, the first cardinality estimation group contains four hosts, and their estimating cardinalities are {3, 3, 4, 5}. The estimating cardinality of the fifth host is 9, which is greater than the upper bound 6, so the host belongs to the next cardinality estimation group, the lower bound of the next cardinality estimation value group is set to 9, and the upper bound is 12. By analogy, the second cardinality estimation group contains four hosts, and their cardinality estimation values are {9, 9, 10, 10}. The third cardinality estimation group contains two hosts, whose estimating cardinalities are {13, 15}, the lower bound of the third cardinality estimation group is 13, and the upper bound is 16.

After AIP is divided into different cardinality estimation groups according to Algorithm 1, the difference in estimating cardinality in each group is reduced. Then start generating sampling IP sequence for each cardinality estimation group, respectively. Using SIP[i] represents sampling IP sequence of the *i*th cardinality estimation group ECG$_i$.

For a host **aip** in ECG, the ideal sampling method is to generate several sampling hosts whose estimating cardinality is **EC(aip)**. However, for a sampling host, we can only determine its real cardinality first and then calculate its estimating cardinality. Moreover, different sampling hosts with the same real cardinality have different estimating cardinalities. Therefore, the peer host set and the real cardinality cannot be determined according to the estimating cardinality of the sampling host.

LE has high estimating accuracy. Therefore, in the actual operation process, the real cardinality of the sampling host can be set with the estimating cardinality as the reference. When generating sampling IP sequence, we need to know three important parameters: sampling point, sampling step, and point sampling number. Each sampling point is an integer value that determines the actual cardinality of the sampling host. A cardinality estimation group contains multiple sampling points. In order to make the estimating cardinalities cover all the cardinality values in the cardinality estimation group, there need to be some sampling points less than the min cardinality **BT** and some sampling points greater than the max cardinality **TP**. The distance between adjacent sampling points is called the sampling step and is recorded as **SS**. In this paper, the equal step size sampling method is used; that is, any two adjacent sampling points have the same step size. The point sampling number refers to the number of sampling hosts generated at each sampling point, which is recorded as **SN**.

According to the definition and requirements of sampling point, **SS** and **SN**, for a cardinality estimation group, the first sampling point is **BT** - **SS**, the second sampling point is **BT**, the third sampling point is **BT** + **SS**, and so on. The *i*th sampling point is **BT + (i-2)** ∗ **SS**.

The last sampling point **L′** needs to be greater than **TP**; it is that (**BT** + (**L'** - (2) ∗ **SS**) > **TP**). After transforming the above formula, **L′** is the minimum integer greater than 2+(**TP**-**BT**)/**SS**. There are **L′** sampling points in a group, and each sampling point has **SN** hosts. Therefore, there are **L'** ∗ **SN** sampling hosts in the sampling IP sequence of a cardinality estimation group. The sampled IP sequence cannot be directly input into the artificial neural network as training data. The next section will introduce how to generate the data used in ANN training.

4.2. Generating Training Data. The sampling IP sequence is composed of the sampling hosts. It can be seen from the previous introduction that a sampling host can be regarded

as a triplet: {IP address **aip**', real cardinality **RC**, and peer host set OIP}. The number of peer hosts in OIP is **RC**. When using an artificial neural network to adjust the result, we need to know the estimating cardinality. To evaluate the estimation error, the sampling host **aip**' needs to generate the cardinality estimation value in the same way used by the host in ANet.

In a time window, for a host **aip** in ANet, the IP address pair composed of **aip** and its peer host **bip** is used to update the LEA. At the end of the time window, find α LEs corresponding to **aip** in LEA and combine them by bitwise "**AND**" operation to obtain the union LE, which is recorded as **ULE(aip)**. Then, the **aip**'s estimating cardinality **EC** is calculated according to **ULE(aip)**. There is a deviation between **aip**'s real cardinality **RC** and the estimating cardinality **EC**, namely, **RC**-**EC**. Since **RC** is unknown, the deviation of estimation is also unknown. In the actual process, all what can be acquired are **EC** and ULE(**aip**).

The purpose of N2CE is to predict and estimate the deviation according to **EC** and **ULE(aip)**. To predict the estimation deviation, we first need to know the relationship among **EC**, **ULE(aip)**, and the estimation deviation. The key to learning this relationship is the hosts whose **ULE(aip)** and **RC** are given. For a sample host **aip**', its **RC** is given. The **ULE(aip')** and **EC** could be calculated based on LEA. Before introducing how to acquire **ULE(aip')** and **EC**, we first give some definitions.

For a host **aip**, the vector [**EC, ULE(aip)**] is called the estimation attribute. The estimation attributes of multiple hosts form an estimation attribute set. When all these hosts come from the sampled IP sequence, they form a training attribute set, which is recorded as **trainX**; when all these hosts come from the ANet, they form a prediction attribute set, which is recorded as **predX**. The sampling host can calculate the estimation deviation Δ according to **RC** (determined when generating the sampling host) and **EC**, where Δ = **RC**-**EC**. The set composed of the estimated deviations of all sampling hosts in a sampling IP sequence is recorded as **predY'**. The estimation deviation of hosts in ANet needs to be predicted by the trained artificial neural network. **predY**$'$ is a set of predicted estimation errors of hosts in a cardinality estimation group. N2CE uses **trainX** and **predY**$'$ to train the ANN and uses **predX** as the input of the trained ANN, and the output result, that is, the estimation error of prediction, is saved in **predY**'.

It can be seen from the process of estimating the cardinalities of hosts in ANet that generating **ULE(aip)** is the key to estimating the cardinality. Generating **ULE(aip)** requires only three types of data: the IP address **aip**, the set of **aip**'s peer hosts, and LEA. For each sampling host, these three types of data are available. Therefore, the same process as estimating cardinalities of hosts in ANet can be used to generate the ULE of the sampling host and estimate the cardinality of the sampling host. That is, first update the LEA using **aip** and **OIP(aip)**, then find α LE from the LEA, and then unionize these LEs to obtain **ULE(aip)**.

When using this method to generate the ULE of a sampling host, the update of LEA by each sampling host cannot affect the update of LEA by other sampling hosts. This is because the estimation error of hosts in ANet is not related to the sampling host. Otherwise, the ULE generated by other hosts will be affected by the previous sampling hosts and cannot accurately reflect the estimation error of hosts in ANet. Therefore, when using this method to generate the ULE of the sampling host, we need to copy an LEA for each sampling host. For high-speed networks, LEA is usually set very large, up to hundreds of megabytes. Copying LEA once for each sampling host will waste lots of computing time and memory. Therefore, this paper proposes an algorithm that can generate estimated host ULE without copying LEA, as shown in Algorithm 2.

For each sampling host sip, Algorithm 2 first obtains α LE of sip in LEA and combines the α LE by bitwise "AND" operation to obtain the union LE ULE. The ULE is then updated with each peer host sip. This method, which combines and generates ULE first and then updates ULE by peer hosts, can obtain the same results as the method, which updates LEA first and then combines and generates union LE. In Algorithm 2, each sampling host only needs g additional bits (LE is composed of g bits) without copying LEA separately. In terms of calculation times, each peer host in Algorithm 2 only needs to update ULE once, while the method, which updates LEA first as what hosts in ANet do, needs to update LE α times. In terms of memory occupation and calculation times, Algorithm 2 can generate cardinality error training data set more efficiently. After generating the training data set, the artificial neural network can be used to learn the relationship among the estimating error, the estimating cardinality, and the **ULE**.

4.3. Training Artificial Neural Network and Predicting the Estimation Error. Artificial neural network learns the relationship between data attributes and estimating error by training data sets. As can be seen from the previous subsection, the data attributes generated by a sampling host include estimating cardinality and ULE, where the estimating cardinality is an integer, and the ULE is a vector composed of g bits. For high-speed networks with huge hosts, g will be set very large, which can be up to thousands or even tens of thousands. If the ULE is directly input into the neural network as part of the training data, too many input attributes will lead to dimension explosion and increase the training time. To reduce the number of input attributes, the convolution method in image processing is used to reduce the dimension of ULE. Figure 3 depicts the neural network model used by N2CE for error prediction.

In the artificial neural network model, the first layer is the input layer, which contains estimating cardinality and g bits of ULE. The second layer is the convolution layer, which is used to convolute ULE. In the convolution layer, the one-dimensional discrete convolution method is used. The convolution kernel used in the layer is an integer vector. The length of the convolution kernel and the moving step during convolution control the dimension of the output attribute. Several hidden layers follow the convolution layer, and each hidden layer contains a dropout layer. The purpose of setting the dropout layer is to reduce overfitting by reducing the

(i) Input: SIP , LEA , g//The length of LE
(ii) H1//hash function used in LE, used to map a host to a bit in LE
(iii) Output: Train data set: **trainX**, **predY′**
(1) For each **{sip, RC, OIP}** in SIP
(2) LEset = **{le$_1$, le$_2$,. . ., le$_\alpha$}** ← Locate α LE related to sip in LEA ;
(3) ULE ← Unionize every LE in LEset by bitwise "AND" operation ;
(4) For each **bip** in OIP
(5) i ← H1(**bip**)
(6) Set the ith bit in ULE to 1
(7) Endfor
(8) **EC** ← calculate the estimating cardinality of **aip**' according to ULE
(9) Insert [**EC**, **ULE**] into **trainX**
(10) Insert (**RC-EC**) into **predY′**
(11) Endfor
(12) Return **{trainX**, **predY}**

Algorithm 2: Generate training data.

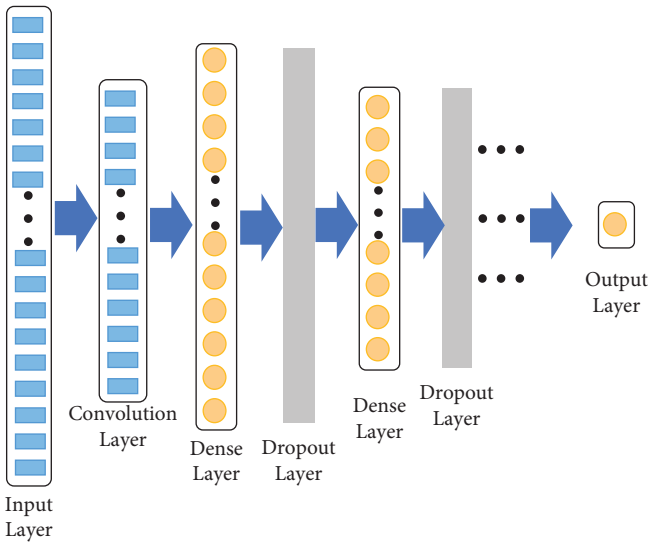

Figure 3: Artificial neural network model.

synergy of hidden nodes. The purpose of N2CE is to predict the estimating error; hence, the final output is a number. Therefore, at the end of the model is the output layer composed of one node.

The learning accuracy of artificial neural network depends on the data and network model used. The data of N2CE is obtained by sampling, and the network model needs to be set before the algorithm runs. Generally, the deeper the network, the more the hidden nodes, the more complete the connection relationship, the higher the accuracy, and the longer the time of using the algorithm. For the N2CE algorithm, the model needs to be trained at the end of each time window, so the network depth and the number of hidden nodes cannot be set too large. Moreover, with the increase of network depth and the number of hidden nodes, overfitting is easy to occur, which reduces the accuracy. In the later experimental part, we will show that high accuracy can be obtained by using only three layers in the neural network.

According to the model in Figure 3, the ANN is trained with the training data set generated by sampling IP sequence, and the ANN learns the relationship between estimation error and data attributes through training. Then the trained artificial neural network can be used to predict the estimation error of the host in ANet.

The data attributes used by N2CE include estimating cardinality and ULE. For the hosts in ANet, these two attributes can be obtained when estimating the cardinality and saved in **predX**. For each host in ANet, the trained artificial neural network can give its estimation error prediction, as defined in the following.

Definition 5. Error prediction. For a host with cardinality equal to **RC**, if the estimating cardinality is **RC**, then the error prediction of the estimation is the difference between **RC** and **EC**, denoted by Δ and Δ = **RC-EC**

According to the definition of Δ, we can acquire the modified estimating cardinality **RC′** by adding the origin estimating cardinality with Δ, **RC'** = **EC** + Δ. N2CE uses an artificial neural network algorithm to reduce the error caused by random factors and make the estimation results more accurate. Next, we illustrate the effect of N2CE on improving the accuracy of cardinality estimation through real-world traffic data.

5. Experiment and Analysis

To evaluate the performance of N2CE on improving the accuracy of cardinality estimation, two groups of high-speed network traffic are used in this paper. There are many famous network traffic data, such as IPtas[25], Caida [26], and Wide [27]. In order to facilitate the experiment, this paper uses Wide traffic data to analyse the performance of different algorithms. The duration of network traffic is 600 seconds, 1800 seconds, and 3600 seconds, starting from 13:00 on May 9, 2018, and 13:00 on April 9, 2019, respectively. The two types of traffic on different days are represented by wide 20180509 and wide 20190409. The summaries of the different time windows of these two types of traffic are listed in Table 1.

Table 1: Information of data in different time windows.

Traffic	Time window(s)	Packets number	Number of hosts in ANet	Number of hosts in BNet
Wide 20180509	600	157858284	146978	1675748
	1800	500028513	366551	4844774
	3600	1008076335	660418	9110573
Wide 20190409	600	164246725	137477	1723667
	1800	462701592	339660	4905480
	3600	889612866	609485	9154056

Table 1 lists the number of packets and the number of hosts in ANet and BNet. From Table 1, we can see that the number of packets and hosts increases with the length of the time window. N2CE calculates multiple hosts in ANet and gets the estimating accuracy by the estimating result of several hosts. The host cardinality is used to reflect the network connection and to monitor the change of the connection in real time. Therefore, the length of the time window should not be set too large. In this paper, the length of the time window is set to 10 minutes, 30 minutes, and 60 minutes, which can reflect the accuracy of the algorithm under different time window.

This paper uses the TensorFlow library to develop the artificial neural network part of N2CE. The activation function is important to the performance of the neural network [28]. The "ReLU" of TensorFlow is used as the activation function in this paper. The program runs on Ubuntu operating system, and the graphics card is NVIDIA GTX 950m with 2G.

The main function of N2CE is to improve the estimation accuracy. The estimation bias (**RC**-**EC**) can reflect the accuracy of the estimation. Therefore, this experiment will analyse the results from the perspective of estimation bias. First, we define several indicators related to estimation bias. In these definitions, AIP is the set of hosts in ANet, $\|AIP\|$ is the number of hosts in AIP, rc_{aip} is the real cardinality of a host **aip** in AIP, and ec_{aip} is the estimating cardinality of a host **aip** in AIP.

Definition 6. Bias rate: $\text{biaRate}_{\text{aip}} = rc_{\text{aip}} - ec_{\text{aip}}/rc_{\text{aip}}$.

Definition 7. Average bias: $\text{avgBia} = \sum_{aip \in AIP}(\text{biaRate}_{\text{aip}})/||AIP||$.

Definition 8. Average absolute bias: $\text{avgAbsBia} = \sum_{aip \in AIP}(\text{biaRate}_{\text{aip}})/||AIP||$.

Definition 9. Standard deviation of bias: $\text{biaStd} = \sqrt{\sum_{aip \in AIP}(\text{biaRate}_{\text{aip}} - \text{avgBia})^2}/||\text{AIP}||$.

Definition 10. Standard deviation of absolute bias: $\text{absBiaStd} = \sqrt{\sum_{\text{aip} \in \text{AIP}}(\text{biaRate}_{\text{aip}} - \text{avgAbsBia})^2}/||\text{AIP}||$.

Bias rate represents the ratio of estimation bias (**RC**(**aip**)-**EC**(**aip**)) and real cardinality. Bias rate removes the influence of the real cardinality on the estimation error and can compare the estimation deviation of different cardinalities. A good estimation should be close to the real cardinality; that is, the bias rate should be close to 0. A good cardinality estimating algorithm should also make the average bias close to 0. However, not all cardinality estimation algorithms with an average bias close to 0 have high estimating accuracy. For different hosts in ANet, the estimating cardinality may be higher or lower than the real cardinality. Therefore, the estimation deviation may be positive or negative. When averaging the estimation deviation, the negative estimation deviation will offset the positive estimation deviation and reduce the average value of the estimation deviation. Therefore, we also use absolute bias, that is, the absolute value of the deviation, to compare the accuracy of the estimation results. Calculate the mean and variance of the absolute bias different hosts to obtain the average absolute bias and standard deviation of absolute bias. This experiment compares and analyses the accuracy of different algorithms by comprehensively using four indexes: average bias, average absolute bias, standard deviation of bias, and standard deviation of absolute bias.

Table 2: Hidden layers of CNN.

Network layer	Hidden nodes	Dropout rate	Activation function
Layer 1	129	0.1	ReLU
Layer 2	256	0.1	ReLU
Layer 3	64	0.2	ReLU

In this experiment, the N2CE algorithm adopts a 4×2048 LEA; each LE contains 1024 bits; the length of the CEG is 100, the sampling step of the sampling IP sequence corresponding to each CEG is 5, and the number of point samples is 10. In the artificial neural network, when convoluting ULE, the convolution kernel length is 8. After the convolution layer, the ANN consists of the following hidden layers: a dense layer composed of 129 nodes using the Rectified Linear Unit (ReLU) as the activation function; a dropout layer with rate 0.1; a dense layer composed of 256 nodes using ReLU; a droop layer with rate 0.1; a dense layer composed of 64 nodes using ReLU; a dropout layer with rate 0.2. When training the artificial neural network, 20 iterations are carried out, and each iteration includes 10 epochs. Table 2 summarizes the structure of hidden layers.

In the experiment, N2CE is compared with DCDS, VBFA, and GSE algorithms, respectively. Let LAA represent the result of N2CE without using the ANN to modify the estimation result, and LAA_DNN represent the result of N2CE with ANN revision. The experimental results are shown in Figure 4, 5, and 6. The ordinates in the picture are average bias, average absolute bias, standard deviation of bias, and standard deviation of absolute bias, respectively, and the abscissa is the number of iterations.

It can be seen from the experimental results that, in different time windows, the value of average absolute bias is higher than that of average bias. This is because the positive

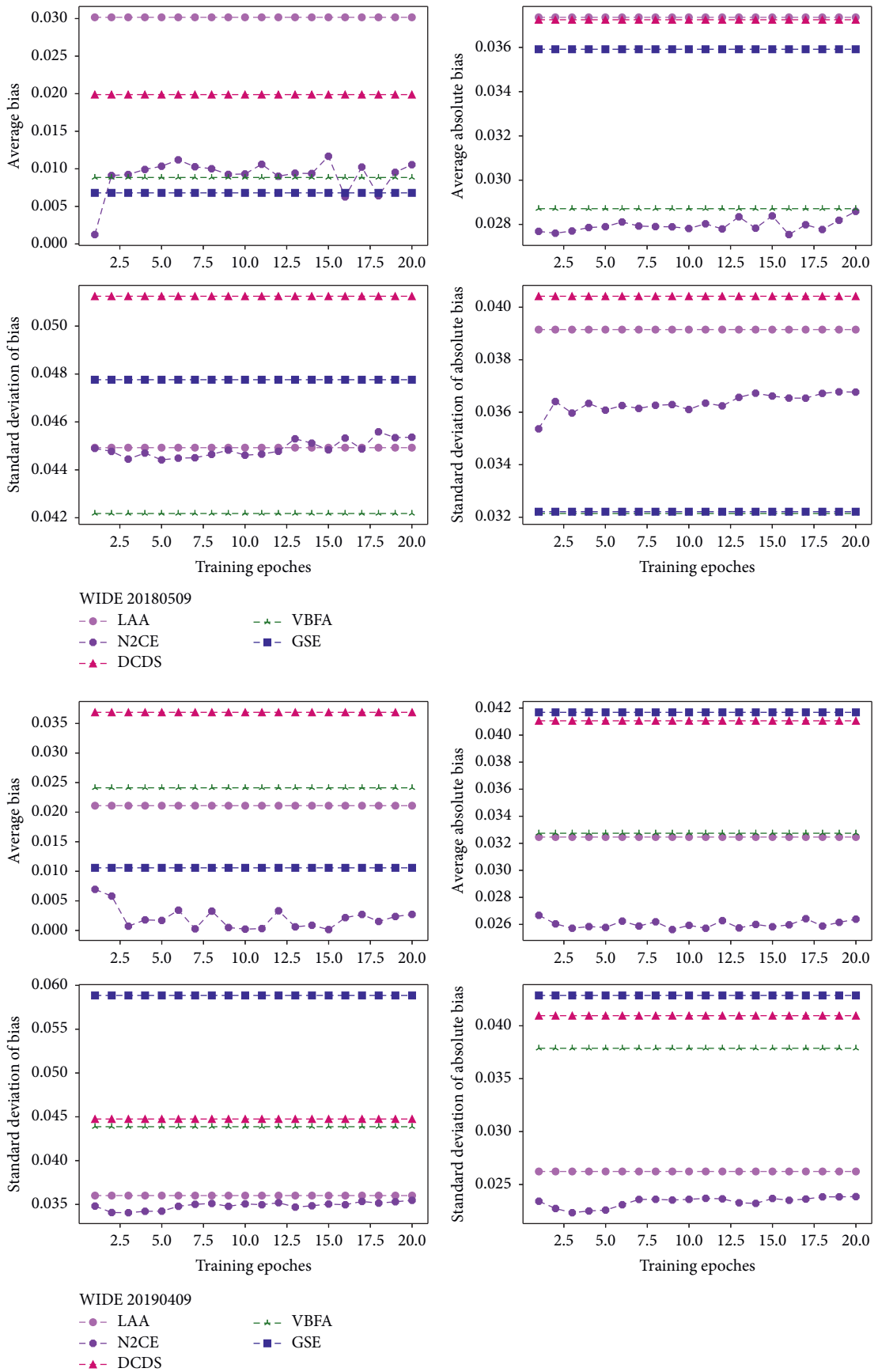

Figure 4: Comparison of estimating accuracy of different algorithms under time window of 600 seconds.

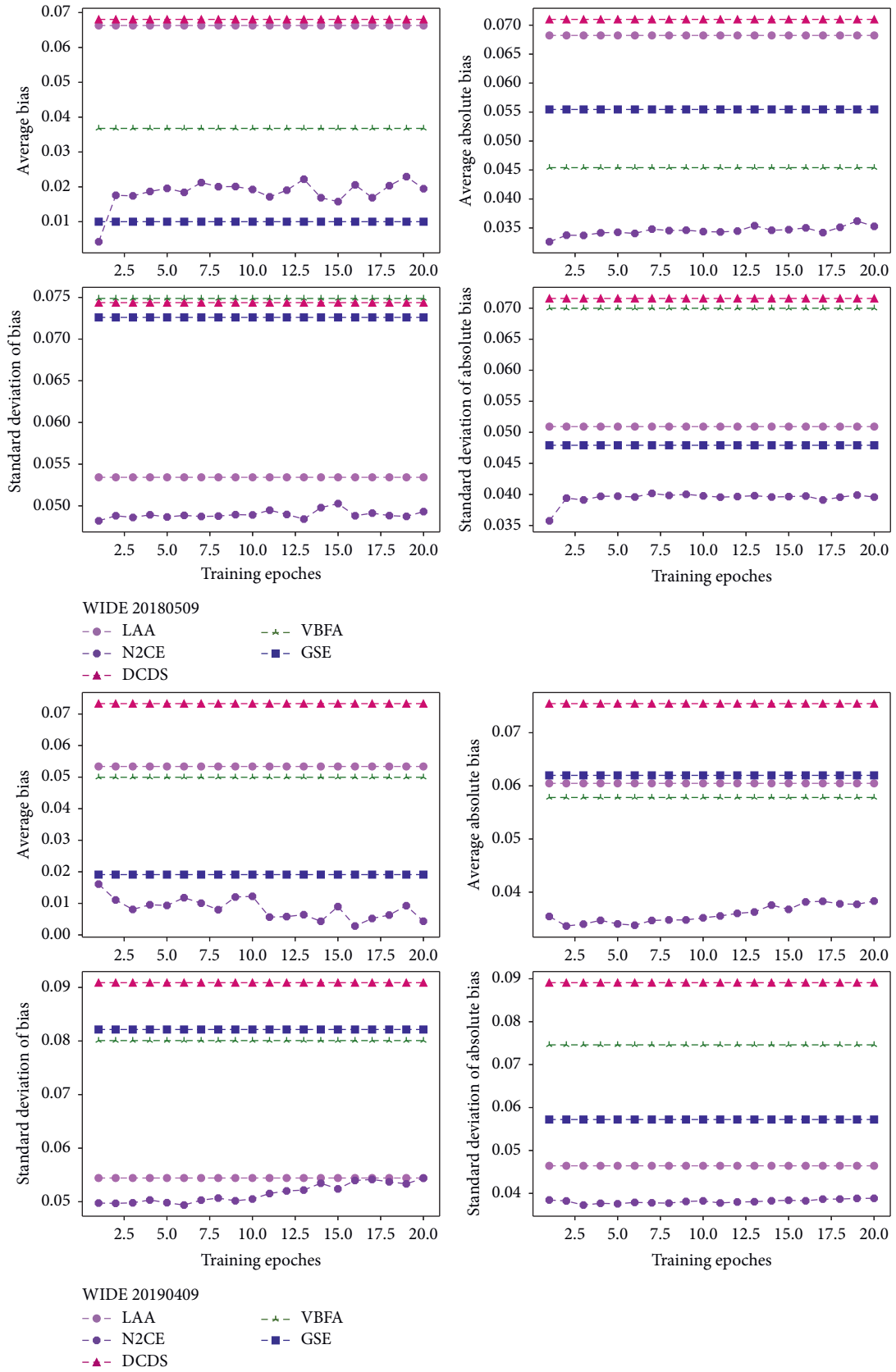

Figure 5: Comparison of estimating accuracy of different algorithms under time window of 1800 seconds.

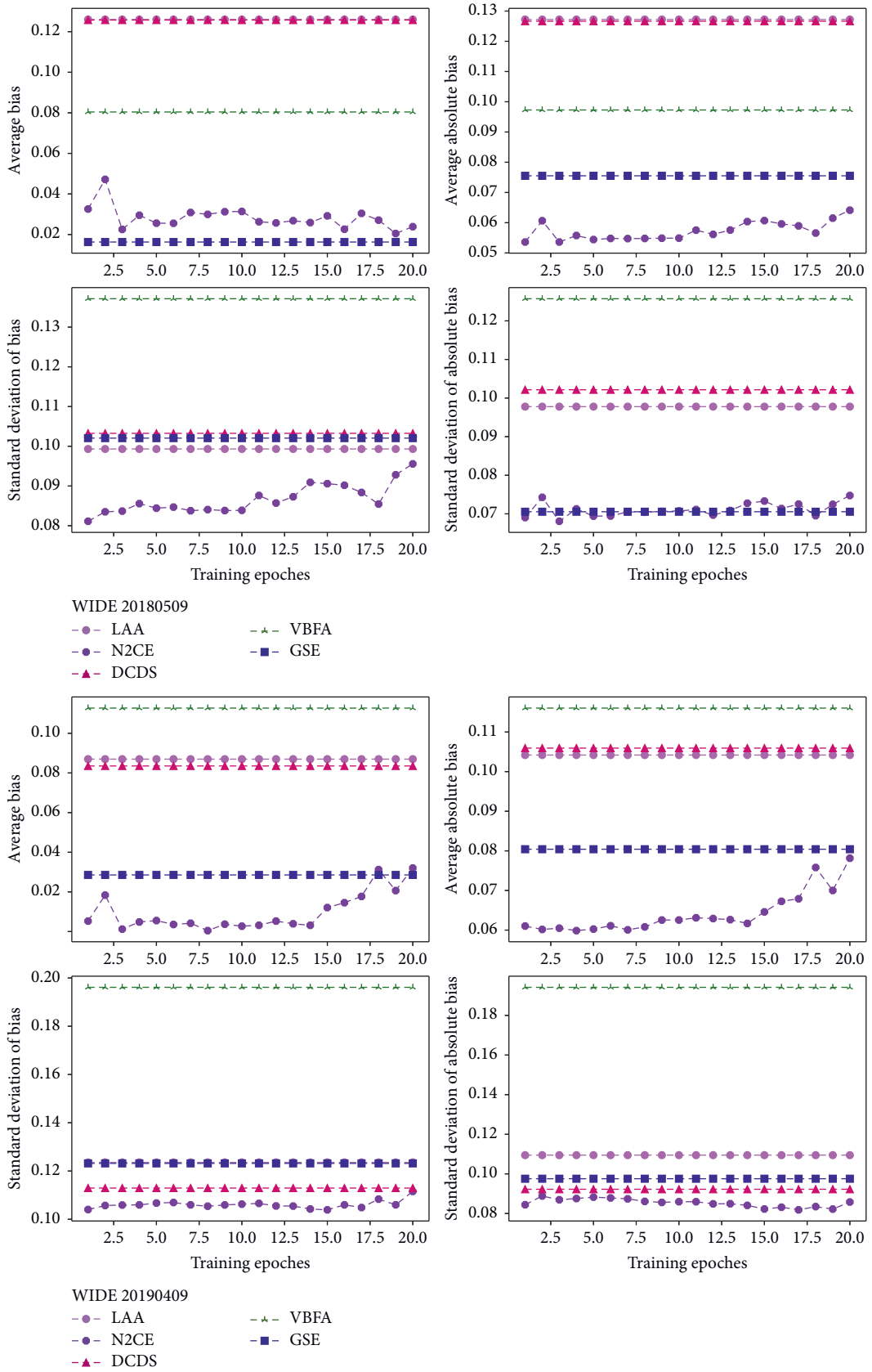

Figure 6: Comparison of estimating accuracy of different algorithms under time window of 3600 seconds.

deviation and negative deviation offset each other in average bias. When the length of the time window increases, the accuracy will reduce. This is because there are more packets and hosts when the time window increases. Between the comparing algorithms, GSE has the lowest average bias, but its average absolute bias is not the lowest. This shows that the estimated value of the GSE algorithm fluctuates up and down around the real cardinality, and the range is large. This phenomenon can be found through the standard deviation of bias analysis of GSE. Contrary to the mean of bias rate, the standard deviation of bias rate is higher than standard deviation of absolute bias rate. This is because the absolute value of bias rate reduces the range of error and reduces the fluctuation.

N2CE adds artificial neural network technology to LAA. It can be seen from the experimental results that the four accuracy indexes of N2CE are better than that of LAA. In N2CE, the average absolute bias decreases by more than 20%. Among all algorithms, N2CE has the lowest average absolute bias. This shows that the estimating cardinalities of N2CE are closer to the real cardinalities than those of other algorithms and have the highest estimation accuracy.

6. Conclusion

The N2CE proposed in this paper uses the technology of artificial neural network to predict the estimation error, to improve the accuracy of cardinality estimation. Generating training data sets is the key to machine learning. N2CE groups the hosts according to the estimating cardinalities and generates sampling IP sequence for each cardinality estimation group. Then, the training data set is generated by combining the linear estimators and updating the peer hosts. According to the training data set generated by the sampling IP sequence, N2CE trains the artificial neural network, predicts the cardinality estimation error, and adjusts the estimation results. Experiments show that N2CE can obtain higher accuracy than existing algorithms. The N2CE can now only run on a single node. However, with the expansion of network scale, a network may have multiple edge routers, which requires a distributed hosts' cardinalities estimation algorithm. In future work, we will study how to aggregate the data of multiple network nodes and improve the accuracy of distributed cardinalities estimation by using N2CE.

Data Availability

The data used in this paper include network traffic, which could be downloaded from the following Wide website: http://mawi.wide.ad.jp/mawi/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the project of Jiangsu Provincial Department of Education (20KJB413002); the Science and Technology Research Project of Jiangsu Provincial Public Security Department (2020KX007Z); the Internet Basic Behaviour Measurement and Analysis: Internet Basic Behaviour Indicator System and Measurement Method (2018YFB1800202); the "Jiangsu Police Institute High Level Talent Introduction Research Start-Up Fund" (JSPIGKZ, JSPI20GKZL404, 2911121350, and 2911121110), and the 2021 Doctor of Entrepreneurship and Innovation in Jiangsu Province (JSSCBS20210599).

References

[1] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: saving (DC)TCP for high-speed extremely shallow-buffered datacenters," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 489–502, 2021.

[2] Q. Xiao, S. Chen, Y. Zhou et al., "Cardinality estimation for elephant flows: a Compact solution based on virtual register sharing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3738–3752, 2017.

[3] R. Saikkonen and E. Soisalon-Soininen, "Cache-sensitive memory layout for dynamic binary trees," *The Computer Journal*, vol. 59, no. 5, pp. 630–649, 2016.

[4] J. J. Hasbestan and I. Senocak, "A short note on the use of the red–black tree in Cartesian adaptive mesh refinement algorithms," *Journal of Computational Physics*, vol. 351, pp. 473–477, 2017, https://doi.org/10.1016/j.jcp.2017.09.056.

[5] R. Padmashani, S. Sathyadevan, and D. Dath, "BSnort IPS better snort intrusion detection/prevention system," in *Proceedings of the International Conference on Intelligent Systems Design & Applications*, IEEE, Salangor, Malaysia, December 2013.

[6] B. Al-Musawi, P. Branch, and G. Armitage, "BGP anomaly detection techniques: a survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 377–396, 2017.

[7] J. Lee and D. Hong, "Collision resistance of the JH hash function," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1992–1995, 2012.

[8] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: a tutorial," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3039–3071, 2019.

[9] H. Li, Z. Bian, P. Zhang et al., "Application-oblivious L7 parsing using recurrent neural networks," *IEEE/ACM Transactions on Networking*, vol. 28, p. 5, 2020, https://doi.org/10.1109/TNET.2020.3000430.

[10] W. Lucas, C. Hartmann, M. Thiele, D. Habich, and W. Lehner, "Cardinality estimation with local deep learning models," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '19). Association for Computing Machinery*, New York, NY, USA, July 2019.

[11] R. Stanojevic, M. Nabeel, and T. Yu, "Distributed cardinality estimation of set operations with differential privacy," in *Proceedings of the 2017 IEEE Symposium on Privacy-Aware Computing (PAC)*, pp. 37–48, Washington, DC, USA, August 2017.

[12] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *Proceedings of the IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, April 2016.

[13] J. Shan, Y. Fu, G. Ni, J. Luo, and Z. Wu, "Fast counting the cardinality of flows for big traffic over sliding windows," *Frontiers of Computer Science*, vol. 11, pp. 119–129, 2017.

[14] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04). Association for Computing Machinery*, pp. 115–120https://doi.org/10.1145/1028788.1028803, New York, NY, USA, October 2004.

[15] M. Ahmed, D. Agrawal, and A. El Abbadi, "Why go logarithmic if we can go linear? Towards effective distinct counting of search traffic," in *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '08). Association for Computing Machinery*, pp. 618–629, New York, NY, USA, March 2008.

[16] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, 2006.

[17] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, 1985, https://doi.org/10.1016/0022-0000(85)90041-8.

[18] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proceedings of the (eds) Algorithms - ESA 2003. ESA 2003. Lecture Notes in Computer Science*, vol. vol 2832, September 2003.

[19] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, 1990, https://doi.org/10.1145/78922.78925.

[20] P. Wang, X. Guan, T. Qin, and Q. Huang, "A data streaming method for monitoring host connection degrees of high-speed links," *IEEE Transactions on Information Forensics and Security*, vol. 6, pp. 1086–1098, 2011.

[21] W. Liu, W. Qu, J. Gong, and K. Li, "Detection of superpoints using a vector Bloom filter," *IEEE Transactions on Information Forensics and Security*, vol. 11, pp. 514–527, 2016.

[22] M. Yoon, T. Li, S. Chen, and J. K. Peir, "Fit a Compact Spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking*, vol. 19, pp. 1253–1264, 2011.

[23] Z. Yang, E. Liang, A. Kamsetty et al., "Deep unsupervised cardinality estimation," *Proc. VLDB Endow.*vol. 13, pp. 279–292, 2019, https://doi.org/10.14778/3368289.3368294.

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

[25] Network IPtas, "Technology Key Labratory of Jiangsu Province, IP Trace and Service," 2021, http://iptas.edu.cn/src/system.php.

[26] CAIDA, "The CAIDA Anonymized Internet Traces," 2019, http://www.caida.org/data/passive.

[27] R. Fontugne, P. Abry, K. Fukuda, D. Veitch, K. Cho, and P. Borgna, "Scaling in internet traffic: a 14 year and 3 day longitudinal study, with multiscale analyses and random projections," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2152–2165, 2017.

[28] W. Lin and G. Chen, "Large memory capacity in chaotic artificial neural networks: a view of the anti-integrable limit," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1340–1351, Aug. 2009.