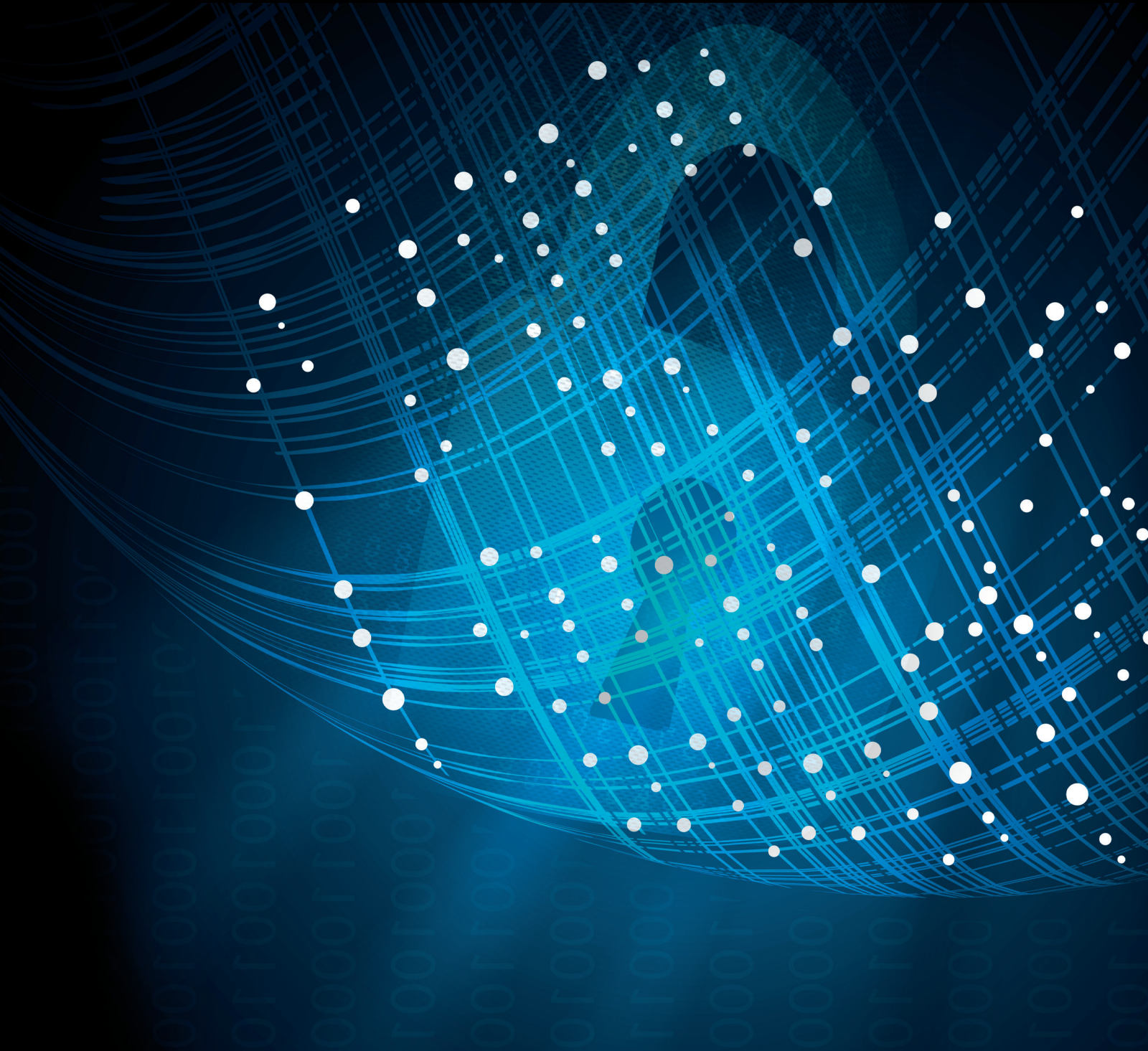


Intelligent Network Traffic Analysis

Lead Guest Editor: Guangjie Liu

Guest Editors: Weiwei Liu, Shujun Li, and Jinwei Wang





Intelligent Network Traffic Analysis

Security and Communication Networks

Intelligent Network Traffic Analysis

Lead Guest Editor: Guangjie Liu

Guest Editors: Weiwei Liu, Shujun Li, and Jinwei Wang







Copyright © 2023 Hindawi Limited. All rights reserved.

This is a special issue published in "Security and Communication Networks." All articles are open access articles distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Chief Editor

Roberto Di Pietro, Saudi Arabia

Associate Editors

Jiankun Hu , Australia
Emanuele Maiorana , Italy
David Megias , Spain
Zheng Yan , China

Academic Editors

Saed Saleh Al Rabae , United Arab Emirates
Shadab Alam, Saudi Arabia
Goutham Reddy Alavalapati , USA
Jehad Ali , Republic of Korea
Jehad Ali, Saint Vincent and the Grenadines
Benjamin Aziz , United Kingdom
Taimur Bakhshi , United Kingdom
Spiridon Bakiras , Qatar
Musa Balta, Turkey
Jin Wook Byun , Republic of Korea
Bruno Carpentieri , Italy
Luigi Catuogno , Italy
Ricardo Chaves , Portugal
Chien-Ming Chen , China
Tom Chen , United Kingdom
Stelvio Cimato , Italy
Vincenzo Conti , Italy
Luigi Coppolino , Italy
Salvatore D'Antonio , Italy
Juhriyansyah Dalle, Indonesia
Alfredo De Santis, Italy
Angel M. Del Rey , Spain
Roberto Di Pietro , France
Wenxiu Ding , China
Nicola Dragoni , Denmark
Wei Feng , China
Carmen Fernandez-Gago, Spain
AnMin Fu , China
Clemente Galdi , Italy
Dimitrios Geneiatakis , Italy
Muhammad A. Gondal , Oman
Francesco Gringoli , Italy
Biao Han , China
Jinguang Han , China
Khizar Hayat, Oman
Azeem Irshad, Pakistan

M.A. Jabbar , India
Minho Jo , Republic of Korea
Arijit Karati , Taiwan
ASM Kayes , Australia
Farrukh Aslam Khan , Saudi Arabia
Fazlullah Khan , Pakistan
Kiseon Kim , Republic of Korea
Mehmet Zeki Konyar, Turkey
Sanjeev Kumar, USA
Hyun Kwon, Republic of Korea
Maryline Laurent , France
Jegatha Deborah Lazarus , India
Huaizhi Li , USA
Jiguo Li , China
Xueqin Liang, Finland
Zhe Liu, Canada
Guangchi Liu , USA
Flavio Lombardi , Italy
Yang Lu, China
Vincente Martin, Spain
Weizhi Meng , Denmark
Andrea Michienzi , Italy
Laura Mongioi , Italy
Raul Monroy , Mexico
Naghme Moradpoor , United Kingdom
Leonardo Mostarda , Italy
Mohamed Nassar , Lebanon
Qiang Ni, United Kingdom
Mahmood Niazi , Saudi Arabia
Vincent O. Nyangaresi, Kenya
Lu Ou , China
Hyun-A Park, Republic of Korea
A. Peinado , Spain
Gerardo Pelosi , Italy
Gregorio Martinez Perez , Spain
Pedro Peris-Lopez , Spain
Carla Ràfols, Germany
Francesco Regazzoni, Switzerland
Abdalhossein Rezai , Iran
Helena Rifà-Pous , Spain
Arun Kumar Sangaiah, India
Nadeem Sarwar, Pakistan
Neetesh Saxena, United Kingdom
Savio Sciancalepore , The Netherlands

De Rosal Ignatius Moses Setiadi ,
Indonesia
Wenbo Shi, China
Ghanshyam Singh , South Africa
Vasco Soares, Portugal
Salvatore Sorce , Italy
Abdulhamit Subasi, Saudi Arabia
Zhiyuan Tan , United Kingdom
Keke Tang , China
Je Sen Teh , Australia
Bohui Wang, China
Guojun Wang, China
Jinwei Wang , China
Qichun Wang , China
Hu Xiong , China
Chang Xu , China
Xuehu Yan , China
Anjia Yang , China
Jiachen Yang , China
Yu Yao , China
Yinghui Ye, China
Kuo-Hui Yeh , Taiwan
Yong Yu , China
Xiaohui Yuan , USA
Sherali Zeadally, USA
Leo Y. Zhang, Australia
Tao Zhang, China
Youwen Zhu , China
Zhengyu Zhu , China





Contents

Research on Network Behavior Risk Measurement Method Based on Traffic Analysis

Qiyao Wang , Xiaolin Zhao , Jiong Guo , Jingfeng Xue , and Bin Zhao 

Research Article (15 pages), Article ID 4501050, Volume 2023 (2023)

A Hybrid Approach by CEEMDAN-Improved PSO-LSTM Model for Network Traffic Prediction

Bilin Shao , Dan Song , Genqing Bian , and Yu Zhao 



Research Article (21 pages), Article ID 4975288, Volume 2022 (2022)

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

Aiping Zhou  and Jin Qian 


Research Article (18 pages), Article ID 3271433, Volume 2022 (2022)

Defending against Deep-Learning-Based Flow Correlation Attacks with Adversarial Examples

Ziwei Zhang  and Dengpan Ye 


Research Article (11 pages), Article ID 2962318, Volume 2022 (2022)

A Lightweight Flow Feature-Based IoT Device Identification Scheme

Ruizhong Du, Jingze Wang , and Shuang Li

Research Article (10 pages), Article ID 8486080, Volume 2022 (2022)

IoT-IE: An Information-Entropy-Based Approach to Traffic Anomaly Detection in Internet of Things

Yizhen Sun, Jianjiang Yu, Jianwei Tian, Zhongwei Chen, Weiping Wang, and Shigeng Zhang 







Research Article (13 pages), Article ID 1828182, Volume 2021 (2021)

Destroy the Robust Commercial Watermark via Deep Convolutional Encoder-Decoder Network

Wei Jia, Zhiying Zhu , and Huaqi Wang

Research Article (11 pages), Article ID 9119478, Volume 2021 (2021)

LightSEEN: Real-Time Unknown Traffic Discovery via Lightweight Siamese Networks

Ji Li , Chunxiang Gu , Fushan Wei , Xieli Zhang, Xinyi Hu , Jiaying Guo , and Wenfen Liu 

Research Article (12 pages), Article ID 8267298, Volume 2021 (2021)

Website Fingerprinting Attacks Based on Homology Analysis

Maohua Guo  and Jinlong Fei 

Research Article (14 pages), Article ID 6070451, Volume 2021 (2021)

AGG: A Novel Intelligent Network Traffic Prediction Method Based on Joint Attention and GCN-GRU


Huaifeng Shi , Chengsheng Pan , Li Yang , and Xiangxiang Gu 

Review Article (11 pages), Article ID 7751484, Volume 2021 (2021)


A Hybrid Association Rule-Based Method to Detect and Classify Botnets

Yuanyuan Huang , Lu Jiazhong , Haozhe Tang , and Xiaolei Liu 

Research Article (9 pages), Article ID 1028878, Volume 2021 (2021)



RT-SAD: Real-Time Sketch-Based Adaptive DDoS Detection for ISP Network

Haibin Shi, Guang Cheng , Ying Hu, Fuzhou Wang, and Haoxuan Ding

Research Article (10 pages), Article ID 9409473, Volume 2021 (2021)

Research Article

Research on Network Behavior Risk Measurement Method Based on Traffic Analysis

Qiyao Wang , Xiaolin Zhao , Jiong Guo , Jingfeng Xue , and Bin Zhao 

School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

Correspondence should be addressed to Jingfeng Xue; xuejf@bit.edu.cn

Received 30 December 2021; Revised 27 June 2022; Accepted 24 November 2022; Published 24 April 2023

Academic Editor: Chien Ming Chen

Copyright © 2023 Qiyao Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

At present, the network security problem is facing a serious threat, and network security events continue to occur. It has become an important link to prevent network attacks and ensure network security. According to the network security protection measures and security technical requirements, it has become an urgent need to establish appropriate security measurement methods and strengthen the monitoring and analysis of network security status. This study proposes a network behavior risk measurement method based on traffic analysis to accurately and objectively evaluate the security state of the network. Traffic is the most basic behavior of the network and the basis of security risk measurement. Firstly, we regard the traffic data as network behavior to build scenarios. Through differential manifold modeling, the traffic data and topology of the network system are semantically described to form a matrix. Then, after manifold dimensionality reduction, the objective risk assessment value can be obtained by manifold mapping and Riemann metric. In this study, the differential manifold theory is applied to network behavior risk measurement, and the innovation of differential manifold in the field of network behavior risk measurement is given. After giving the network behavior risk measurement theory, we first verify the effectiveness of the proposed method through the simulation experiments. Secondly, the public CIC-IDS-2017 data set is used for analysis and calculation to prove the accuracy of the proposed method.

1. Introduction

1.1. Background. At present, computer network is playing a more and more important role. However, the potential threat of computer network is its own vulnerability and the vulnerability of communication equipment. On the one hand, computer network hardware and communication equipment are vulnerable to the influence of natural environmental factors such as dust, humidity, temperature, electromagnetic field, and man-made physical damage [1]. On the other hand, due to the nature of information sharing and open platform of the network itself, important assets, software resources, and data information in the computer are vulnerable to illegal theft, replication, tampering, and destruction. All these lead to the damage, loss, and security accidents of assets and data information in the computer network system. Network behavior risk assessment can judge the security performance of the network to a certain extent. On this basis, it can continuously improve the

security of the network for the weak links of the network and further improve the network security situation [2]. Therefore, it is very important to propose effective network security measurement methods and improve them. With the rapid development of computer network, traffic attacks appear frequently which is the most common type of network attacks [3]. Therefore, it is also very important to propose network security metrics for traffic attacks.

At present, the research on network security measurement is mainly divided into two perspectives: network management security and network technology security. The research of network management security mainly focuses on the published network security guidelines or international standards [4]. Network technology security measurement is mainly divided into qualitative measurement and quantitative measurement [2, 5]. Qualitative measurement generally does not use mathematical methods. The evaluator can directly draw a conclusion on the evaluation network through expert

experience and existing knowledge and through the inductive analysis of the current network security situation [6]. Quantitative measurement makes a quantitative judgment on the network security situation by constructing a mathematical model or through a certain quantitative method [7–10]. Compared with the subjective judgment of qualitative evaluation, quantitative evaluation can effectively quantify the network security situation, but it requires a lot of data analysis and comparison, and the implementation is more complex [6, 11]. Both of them have the problems of weak objectivity and inaccurate measurement.

1.2. Innovation. This study summarizes and analyzes the common methods of network security measurement. Aiming at the problems of weak objectivity and inaccurate measurement in the existing methods, this study puts forward the network behavior risk calculation method, obtains the network attack and defense behavior through the traffic analysis, and uses the differential manifold theory to describe the network behavior state in the attack and defense process, so as to measure the network behavior risk. Finally, based on the measurement model proposed in this study, the feasibility and effectiveness of the algorithm are verified by attack and defense experiments in real environment and public CIC-IDS-2017 data set.

The network behavior risk measurement method based on traffic analysis proposed in this study can reflect the security state of the network, intuitively analyze the change degree of the security state of the network system in the process of attack and defense, and accurately evaluate the performance of the network.

The innovation of this study is as follows:

- (1) The network traffic is regarded as network behavior, and the definition of network behavior risk is proposed to realize the quantitative analysis of network behavior
- (2) The local linear embedding dimension reduction algorithm of manifold learning is used to reduce the dimension of the index
- (3) Aiming at the problems of weak objectivity and inaccurate measurement in the existing methods of network security measurement by describing the network behavior state in the process of attack and defense, the differential manifold theory is applied to the network behavior risk measurement, and the network behavior risk calculation method is proposed

2. Related Research

Network technology security is the evaluation and analysis of the system security of the network, such as availability, integrity and confidentiality, and the vulnerability risk of the network. Network technology security measurement is mainly divided into qualitative measurement and quantitative measurement [2, 6].

Sheng et al. [12] introduced analytic hierarchy process, according to the characteristics of network architecture defined by software, selected several typical indicators affecting network security status, and calculated the overall network security value. Wang et al. [13] made full use of the advantages of grey analytic hierarchy process to evaluate the network security risk. The qualitative measurement model has the advantages of convenient evaluation and strong applicability, but human factors have a great impact on the final evaluation results, lack of objectivity, and often have the problem of inaccurate evaluation [6].

The network security measurement method based on attack graph is a common quantitative measurement method. Phillips and Swiler [14] mapped the network system into an attack graph for the first time, intuitively displayed and analyzed the possible attack paths, vulnerabilities, and important nodes in the network, and proposed a new network security measurement algorithm based on these attacks information. Literature [15] combined with attack graph technology used Bayesian network to determine the atomic attack nodes of the network, so as to obtain the overall security value of the network and optimize the measurement results. However, the network security measurement method based on attack graph is not objective and is not suitable for complex networks.

The detection of network traffic data can also reflect network anomalies. The latest trend of network anomaly detection based on network traffic data includes emerging machine learning technologies such as artificial neural network (ANN), support vector machine (SVM), 1-nearest neighbor (KNN), decision tree, clustering, and statistics [16]. In related research, traffic classification is the first step to identify malicious use of network resources by anomaly detection and other activities [17]. Wang et al. [18] proposed a malware traffic classification method based on convolutional neural network and taking traffic data as image for security detection. Marir et al. [19] used a group of multi-layer support vector machines and deep feature extraction in large-scale networks to identify abnormal behaviors and detect network security. First, the distributed deep trust network is used to nonlinear reduce the dimension of network traffic data, and then the extracted features are used as input to construct multilayer support vector machine through spark-based iterative dimension reduction paradigm. Shubair et al. [20] proposed an intrusion detection system based on traffic data, which takes advantage of the combination of KNN method and fuzzy logic. The minimum mean square method is used for error reduction, KNN selects the best matching class, and fuzzy logic selects the flow class label. Liu et al. [10] proposed a detection method based on Riemannian measurement of traffic data, which uses fast Fourier transform and information entropy to detect attacks.

With the continuous development of network security performance requirements, the previous measurement methods are difficult to meet the needs of measurement accuracy and accuracy, and the introduction of mathematical principles can describe the network security situation with more objective and accurate methods and values

[21, 22]. In this study, the differential manifold theory is applied to network behavior risk measurement. Differential manifolds have been widely used in theoretical physics and high-dimensional data dimensionality reduction research [23]. With the rapid development of network informatization, more and more differential manifolds have been applied in the field of computer networks [24]. For example, the differential manifold and Riemann metric are applied to the robot performance and network control simulation to improve the robot operation and motion performance. In the field of computer vision, introducing differential manifold for information extraction [1, 25] and using differential manifold to improve image processing efficiency [26]. Using differential Manifolds and Riemann metrics to study network attack and defense effectiveness and to achieve the evaluation of network system security performance [27, 28]. Therefore, analyzing network risk and evaluating cyber security situation through mathematical principles have gradually become the trend of measurement research.

To sum up, the comparison of various common network security measurement methods is shown in Table 1.

Aiming at the problems of weak objectivity and inaccurate measurement in the existing network security measurement methods, this study summarizes and analyzes the common methods, puts forward the network behavior risk calculation method, and applies the differential manifold theory to the network behavior risk measurement by describing the network behavior state in the process of attack and defense. Finally, based on the measurement model proposed in this study, we verify the feasibility and effectiveness of the algorithm by using the public CIC-IDS-2017 data set and conducting attack and defense experiments in real environment.

3. Traffic Behavior Analysis and Differential Manifold

This study proposes the network behavior risk calculation method, obtains the network attack and defense behavior through traffic analysis, uses the differential manifold theory to model the network attack and defense behavior, and makes the network security measurement through the network attack and defense behavior.

3.1. Network Security Measurement. Network security means that valuable assets such as data and information in the network system will not be leaked, tampered with, or damaged due to wrong operation inside the network or malicious attack outside the network. The ideal situation of network security is that the network will not be affected by external attacks. However, the network is always facing threats.

Network security measurement is to detect the vulnerabilities in the network, judge the possible network attack means and several existing network attack paths, and determine the current security state of the network through the evaluation of security data indicators such as vulnerabilities, assets, and traffic in the network [29].

The basic steps of network security measurement are shown in Figure 1.

3.2. Traffic Behavior Analysis. Traffic is the most basic behavior of the network and the basis of security risk measurement. The complete traffic includes the data information of application layer, transport layer, network layer, and physical layer. The definition of traffic [30] on the transport layer is it describes the packet string with the same IP address, port number, and protocol (TCP, UDP, ICMP, and so on).

The behavior analysis of network traffic mainly analyzes the behavior characteristics of traffic by analyzing the characteristic parameters such as bandwidth/throughput and delay. Taking network traffic as the research object, this study takes the characteristic parameters such as bandwidth and delay of traffic as indicators to analyze the behavior of network traffic. The behavior analysis of network traffic is a direct and effective means to obtain the state of the network. It can understand and master the behavior of traffic, help to obtain the characteristics of network performance, reliability, and security, and establish the behavior model of the network.

From the perspective of network traffic data analysis, this study regards the network traffic data collected from the actual network as a network behavior by collecting and monitoring the network packet information in real time and then proves that the network system is a topological manifold, uses the differential manifold to model the network behavior, and measures the network security through the network behavior.

3.3. The Relationship between Network Security Metrics and Differential Manifolds. We use the differential manifold theory to study the network security metrics. It is necessary to prove that the network system is a topological manifold in order to calculate the network risk and judge the network security state by using the differential structure and given Riemannian metric. The specific proof is as follows.

Definition 1. Network topology.

Network topology is the shape of network and the physical connectivity of network. Network topology refers to the physical layout of various devices interconnected by transmission media [31].

From the definition of network topology, it is obvious that the network system is a topological space. Any subsystem of a network system must also be a network system. Several network systems can be connected to form a large network system through topology. Hausdorff space is a topological space, and Hausdorff space is a topological space whose points are "separated by domains." Therefore, the network system is a Hausdorff space.

Definition 2. Topological manifold.

Let m be a Hausdorff space. If there exists an open field $U \in M$ at any point such that the open field is homeomorphic with the open subset in Euclidean space R^n , then M is a topological manifold [31].

TABLE 1: Comparison table of common network technology security measurement methods.

Measurement method	Advantage	Disadvantage
Analytic hierarchy process	Qualitative and quantitative measurement method	Strong subjectivity, incomplete analysis, and poor comparability of measurement results
Attack graph	Mature method and can measure the direction and path of network attack	Weak objectivity and not suitable for complex networks
Machine learning	The current research hotspot and can identify various network attacks	The model relies on a large amount of data training
Flow analysis	Can accurately reflect network anomalies	At the stage of development
Principles of mathematics	Markov chain and differential manifold at el. and can accurately describe network behavior and network security state	Little research on relevant aspects and at the stage of development

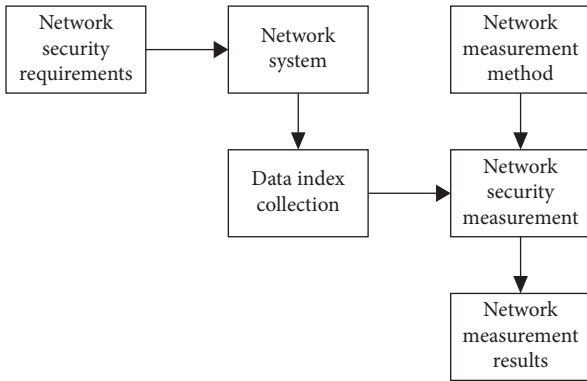


FIGURE 1: Network security measurement process.

According to the definition of topological manifold, if every point in a Hausdorff space can find an open field homeomorphic to a Euclidean space, then the space becomes a topological manifold. For a network system, we can always find a homeomorphism mapping, which satisfies $f: x \rightarrow U(x) \in R_+^n$, and then the network system is a topological manifold.

Definition 3. Riemannian metric.

Let M be a n dimensional optical slip flow shape and construct a positive definite and symmetric second-order covariant smooth tensor field g on M , that is, $g(p)$ is for any $p \in M$ a positive definite, symmetric second-order covariant tensor on T_p^M , then g is a Riemannian metric of M , and M is a Riemannian manifold [32].

3.4. The Feasibility of Measuring Network Security by Differential Manifold. Differential manifold has been widely used in theoretical physics and dimensionality reduction of high-dimensional data [23]. With the rapid development of network informatization, there are more and more applications of differential manifold in the field of computer network [24]. For example, differential manifold and Riemannian metric are applied to the simulation of robot performance and network control to improve the robot operation and motion performance. In the field of computer vision, the information extraction method of differential manifold is introduced to improve the efficiency of image processing [25], differential manifold is used to improve image processing efficiency [26], differential manifold and Riemannian metric are used to study network attack and defense utility, to realize the evaluation of network system security performance [27, 28], and so on.

Network security measurement is an effective process to measure security and protect data based on a certain scale [33]. The network needs to quantify the security elements related to the system security quality, such as vulnerability, risk, attack, and defense [34].

Differential manifold is a mathematical model that can objectively calculate and measure things. The advantage of differential manifold is that it can keep the data topology unchanged in the change of dimension and explore the internal geometric structure and regularity hidden in the

data. Therefore, differential manifolds can be used to measure network behavior risk. The network system itself is regarded as a manifold in one or more scenarios.

Compared with traditional methods, network security assessment based on differential manifold can solve problems that are not objective and comprehensive and can better reflect the impact of changes in security metrics on network security changes, making the metrics more objective and accurate than previous assessment methods [21, 27]. At the same time, the network security evaluation method based on differential manifold can consider the network security risk at multiple levels and explain the transformation of the network security state from the perspective of attack and defense utility.

4. Research on Network Behavior Risk Measurement Based on Manifold

The process of using differential manifold to measure network behavior risk is as follows: first, collect data and reduce the dimension of indicators to obtain a series of measurement indicators. In addition, second, construct a network security measurement index group through these indexes. Third, calculate the network security state value before and after network attack by using network system differential manifold. By comparing the risk values in different time periods, we can judge whether the network is at risk in this period.

4.1. Definition of Network Behavior Risk. Network behavior risk involves key elements such as vulnerability, threat, asset, risk, and so on [35]. The factors involved in network behavior risk are the result of mutual influence and interaction. Vulnerability and threat will increase network risk. Risk mainly affects assets. Security measures to deal with risk can reduce the impact of vulnerability and threat. The relationship between various factors of network behavior risk is shown in Figure 2.

Network security events cause network behavior risk. The occurrence of network behavior risk is a function of the emergence of threats and the utilization of vulnerability. The impact of network behavior risk is the destruction and loss of network assets. Therefore, network behavior risk can be defined as follows:

$$R = f(T, V, A). \quad (1)$$

Among them, R is network behavior risk, f is network behavior risk calculation function, T is network threat, V is network vulnerability, and A is information assets in the network.

4.2. Network Security Baseline. In order to realize the quantitative assessment of network risk, it is necessary to set an objective baseline that can reflect whether the network is safe or not. Network security baseline is the dividing point to judge whether the network is safe or not. By comparing the network security risk status and

network security baseline, we can determine whether the current network is at risk. First, we need to study the security attributes of the network security infrastructure such as assets and information; second, the evaluation standard and calculation model of network security baseline are established; finally, we compare the security baseline to determine the current risk status of the network. At the same time, when the security requirements or security factors in the network system change, the network security baseline should be adjusted appropriately.

As shown in Figure 3, it is assumed that the network security baseline value under certain security factors is α . The result of network behavior risk value calculated after network security detection is β . By comparison, if the results of the two values are consistent or within a certain error range, then it shows that the network system is in a safe state in this period of time. If the risk value is lower than the risk value β much larger than the baseline α . So, it shows that in this period of time, the security state of the network system has changed, and the network may be attacked in a network risk state.

4.3. The Selection of Measurement Index of Network Behavior Risk. The selection of network behavior risk measurement index should have the following characteristics: (1) the index is clear, the meaning of data index is clear, each index is relatively independent, there is no redundancy, and it is easy to calculate; (2) it can cover all kinds of network indicators, including traffic, host, and other common indicators; and (3) it is easy to expand. With the complexity of the network and the changes of other factors, the network behavior risk measurement index can add the necessary data indicators that affect the network security factors.

In this study, the indicators selected in the measurement network are shown in Table 2. Section 6.2 of this study uses CIC-IDS-2017 data set for experiment. Among them, CIC-IDS-2017 data set contains more than 80 characteristic indicators.

4.4. Dimension Reduction of Network Behavior Risk Measurement Index. Network behavior risk measurement is a comprehensive analysis of the existing network security state. Therefore, there are some problems in the measurement: (1) there are many attributes involved in the measurement index and (2) there are many kinds of indicators. Each level collects dozens or even hundreds of indicators. Based on the above two points, after the completion of the index collection, we need to simplify the index. The comparison of common dimensionality reduction methods is shown in Table 3:

To sum up, locally linear embedding algorithm, which is based on manifold learning, can better maintain the original key features and geometric properties of data than other methods. Therefore, this study uses manifold learning method to reduce the dimension of the index.

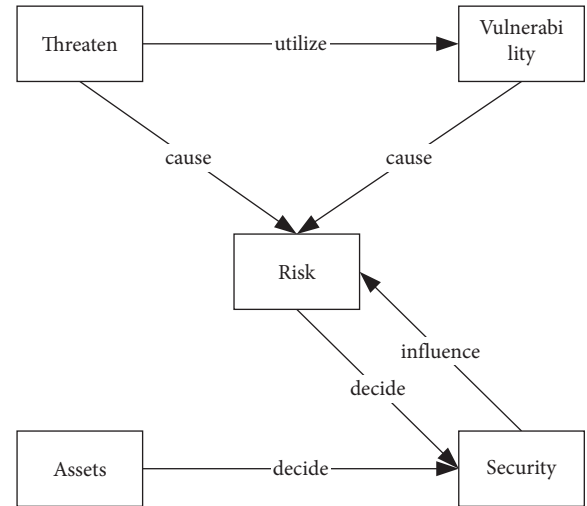


FIGURE 2: The relationship among various factors of network behavior risk.

Locally linear embedding algorithm considers that every data point can be constructed by the linear weighted combination of its nearest neighbors. The main steps of the LLE algorithm are divided into four steps:

- (1) Find k nearest neighbors of each index data point in the original index data sample space
- (2) Approximately calculate the weight matrix M of the index sample space through these nearest neighbors
- (3) Decompose the weight matrix M to obtain eigenvalues and eigenvectors
- (4) Take the eigenvector corresponding to the smallest d eigenvalues, that is, the new index data after dimension reduction

4.5. Network Behavior Risk Measurement. According to the definition of network behavior risk in Section 4.1, this study uses differential manifold to measure and calculate network behavior risk, which can describe network attack and defense process, and describe network scene and network behavior. At the same time, differential manifold can map network system and network index data space into a high-dimensional space to better describe the change degree of network behavior risk. When the network is threatened by external attacks, the state of network indicators will also change to a certain extent. Then, the network security risk value can be expressed by measuring the change value of network indicators through a certain calculation method. The relationship between network risk and index changes is shown in Figure 4.

In Section 4.3, the network risk measurement index group has been established. The vector group composed of these indexes can form a high-dimensional data manifold space. If the high-dimensional data manifold is surface integrated, the calculated results can represent the corresponding network risk state change of the network system when the data index changes. The surface integral of

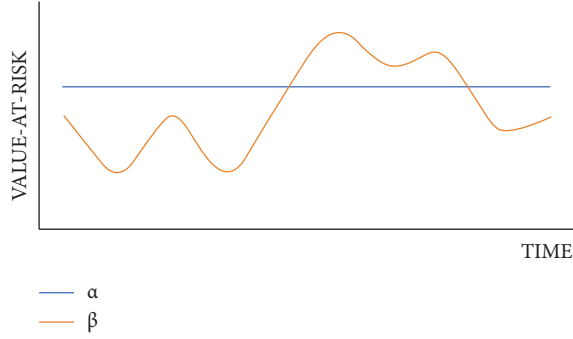


FIGURE 3: Network security status change.

TABLE 2: Network security measurement metrics.

One-level indicators	Two-level indicators	Three-level indicators
Network security risk	Computing storage	CPU utilization
		Memory utilization
		Disk utilization
	Bandwidth consumption	Bandwidth utilization
	Flow change	Instantaneous flow
Flow rate		
Data packet	Data packet	Packet loss rate
		Packet length
		Packet quantity

a manifold is actually the volume of the manifold in a high-dimensional space. Therefore, the results of network behavior risk can be expressed by the hypervolume of the high-dimensional data manifold composed of network security indicators.

In the high-dimensional data manifold space, we can define certain differential structures and Riemannian measures to calculate the changes of these indicators in the network attack and defense. After integral operation, the transformation quantity can represent the risk of the network. If the matrix form is used to represent the current network index state of the network system, assume that the data index set is (x_1, x_2, \dots, x_m) . Then, in time (t_1, t_2, \dots, t_n) , there are $m * n$ state variables in the matrix. Therefore, the network index state matrix S can be expressed as follows:

$$S = \begin{bmatrix} s(x_1, t_1) & s(x_2, t_1) & \cdots & s(x_m, t_1) \\ s(x_1, t_2) & s(x_2, t_2) & \cdots & s(x_m, t_2) \\ \cdots & \cdots & \cdots & \cdots \\ s(x_1, t_n) & s(x_2, t_n) & \cdots & s(x_m, t_n) \end{bmatrix}. \quad (2)$$

In the measurement of network behavior risk, we need to define the measurement function f . After the definition of network risk is given, the change process of network security state is expressed as the change of points and data on the differential manifold. The change state of network system security index constitutes a “point” in the differential

manifold. By integrating the change state of security index on the differential manifold, the risk measurement result is obtained. For matrix S , the process of calculating network risk by using differential manifold and metric function can be expressed as follows:

$$f: R^{n^2} \longrightarrow R_+^n. \quad (3)$$

The metric function f represents the integral function of the differential manifold to calculate the network risk. In the process of network attack and defense, the risk of network security behavior changes instantaneously. There is no reference significance to calculate the value of network security state at a single time. Only in a period of dynamic change can it have the significance of measuring security. Therefore, in a period of time t , when the network index set $X = (x_1, x_2, \dots, x_n)$ changes, the calculation of network risk is expressed by the following formula:

$$R = \int \Delta(S) ds. \quad (4)$$

Through formula (4), the index state in the network system is mapped into a matrix, and the change of the index is combined with the change of the network risk, and the general calculation formula of the network risk is given. In this study, the description of the change of the network index set is based on the differential manifold. The process of the change of the index set is the change process of the high-dimensional manifold of the index data. The integral of the manifold change represents the change of the network security state. Finally, the network risk can be determined by comparing with the security baseline value.

5. Network Behavior Risk Calculation Model

The network behavior risk measurement process can be divided into three parts: data index collection and processing, network measurement manifold construction, and network behavior risk calculation.

5.1. Index Dimension Reduction Calculation Model. In this study, the locally linear embedding LLE algorithm is used to reduce the dimension. The LLE algorithm considers that the original data sample is linear in a small part. Suppose there is a sample x_i . Then, several sample points x can be found in the original high-dimensional neighborhood x_1, x_2, \dots, x_k . It exists as follows:

$$x_i = w_{i1}x_1 + w_{i2}x_2 + \cdots + w_{ik}x_k, \quad (5)$$

where $w_{i1}, w_{i2}, \dots, w_{ik}$ is the weight coefficient. After dimension reduction by the LLE algorithm, part of the linear relationship of data can still be maintained in the new space, and the weight relationship before and after dimension reduction can be kept unchanged or slightly changed. Namely,

$$x_i' = w_{i1}x_1' + w_{i2}x_2' + \cdots + w_{ik}x_k'. \quad (6)$$

TABLE 3: Comparison of dimension reduction algorithms.

Algorithms	Advantage	Disadvantage
PCA	Linear mapping to eliminate the interaction of data	Only deal with the sample variance, not nonlinear data
LDA	Used for big data classification	Not suitable for non-Gaussian distribution samples
KPCA	Nonlinear data can be processed on the basis of PCA	Depend on the choice of kernel function
MDS	Keep sample difference on Euclidean distance	Not consider the distribution and interaction of adjacent data
Isomap	Preserve the geometric properties of samples on manifold distance	Not suitable for manifolds with large curvature
LLE	Solve the problem of high-dimensional data distribution	Need to assume that the manifold of the sample exists

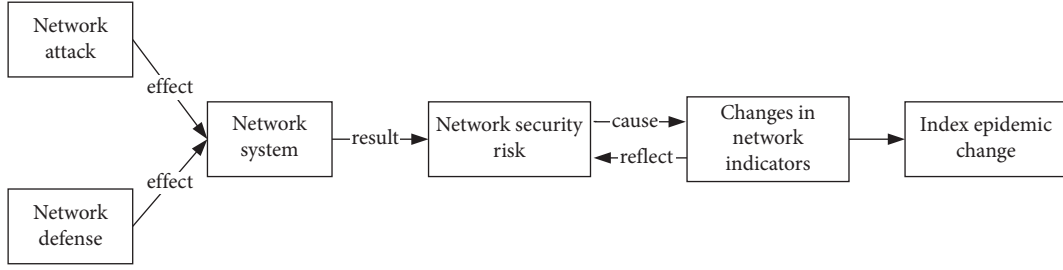


FIGURE 4: Relationship between network risk and network index change.

The main steps of LLE algorithm are divided into four steps as shown in Section 4.4.

In order to illustrate the calculation process, the data space of m n -dimensional samples $X = x_1, x_2, \dots, x_m$ is selected and the loss function is matrixed to obtain as follows:

$$\begin{aligned}
 J(W) &= \sum_{i=1}^m \left\| x_i - \sum_{j \in Q(i)} w_{ij} x_j \right\|_2^2 \\
 &= \sum_{i=1}^m \left\| \sum_{j \in Q(i)} w_{ij} x_i - \sum_{j \in Q(i)} w_{ij} x_j \right\|_2^2 \\
 &= \sum_{i=1}^m \left\| \sum_{j \in Q(i)} w_{ij} (x_i - x_j) \right\|_2^2 \\
 &= \sum_{i=1}^m W_i^T (x_i - x_j) (x_i - x_j)^T W_i,
 \end{aligned} \quad (7)$$

where $Q(i)$ is the k nearest neighbor sample set of i , $W_i = w_{i1}, w_{i2}, \dots, w_{ik}^T$. The weight coefficient satisfies the following equation:

$$\sum_{j \in Q(i)} w_{ij} = W_i^T I_k = 1. \quad (8)$$

Let matrix $Z_i = (x_i - x_j)(x_i - x_j)^T$, calculate the weight coefficient W_i as follows:

$$W_i = \frac{z_i^{-1} I_k}{I_k^T z_i^{-1} I_k}. \quad (9)$$

Suppose that the projection of sample set in low dimension d ($d < m$) is $Y = \{y_1, y_2, \dots, y_m\}$. In order to keep the linear relationship after dimension reduction, the matrix objective loss function is as follows:

$$\begin{aligned}
 J(Y) &= \sum_{i=1}^m \left\| y_i - \sum_{j=1}^m w_{ij} y_j \right\|_2^2 \\
 &= \sum_{i=1}^m \|Y I_i - Y W_i\| \\
 &= \text{tr}(Y(I - W)(I - W)^T Y^T).
 \end{aligned} \quad (10)$$

Let $M = (I - W)(I - W)^T$, then $J(Y) = \text{tr}(YMY^T)$. Then, we can get the new data sample Y after dimension reduction.

To sum up, the specific flow of the LLE algorithm is shown as follows:

Input: sample set $D = x_1, x_2, \dots, x_m$

The nearest neighbor parameter k

After dimension reduction, the dimension of space d

Process:

For $i = 1, 2, \dots, m$ do

Calculate k -nearest neighbors of x_i

Calculate reconstruction coefficient w_{ij}

End for

Obtain the correlation matrix M

Eigen decomposition of matrix M

Returns the eigenvectors corresponding to the minimum d eigenvalues of a matrix

Output: the sample after dimension reduction of the original sample set d

5.2. Data Index Collection and Processing. In order to eliminate the impact of the indicator units and make each indicator have the same impact on the calculation results, it is necessary to standardize the collected network security related indicator data.

Standardization can remove the restrictions of different units and scales between data; that is, different data are scaled as a whole according to a unified scale and converted into pure values in a specific interval. Generally, the min-max standardization method is used to map the original data to $[0, 1]$ interval. For example, a standardized transformation of (11) is carried out for a certain index data as follows:

$$y_i = \frac{x_i - \min_{1 \leq j \leq n} \{x_j\}}{\max_{1 \leq j \leq n} \{x_j\} - \min_{1 \leq j \leq n} \{x_j\}}. \quad (11)$$

So, the new sequence $y_1, y_2, \dots, y_n \in [0, 1]$ is obtained by calculation and is nondimensional, where $\max_{1 \leq j \leq n} \{x_j\}$ is the maximum value of the original data and $\min_{1 \leq j \leq n} \{x_j\}$ is the minimum value of the original data.

5.3. The Construction of Risk Measurement Model of Network Behavior. The change of network security state is a function of the change of network metrics over time. Under the condition of function representation, a set of C^r compatible

total covers can be found in the network topological manifold and the network topological manifold can be constructed as a network differential manifold. In the network differential manifold, we only need to give the corresponding Riemannian metric, and then we can use the differential manifold theory to measure the network behavior risk.

For n -dimensional manifold space, the distance between any two points can be expressed as follows:

$$ds^2 = g_{uv}(x)dx^u dx^v, \quad (12)$$

where $x = (x^1, x^2, \dots, x^n)$ and g_{uv} is a Riemann metric defined in n -dimensional space. Generally, we choose the Riemann metric with symmetric positive definite, $g_{uv} = g_{vu}$, and then ds^2 is the distance calculation method of two points in n -dimensional space under different Riemann metric.

If we use matrix form to describe Riemannian metric, let $g = g_{uv}$, $x = x^\alpha$, and $dx = dx^\alpha$. Thus, the distance formula (12) can be expressed as follows:

$$ds^2 = dx^T g dx. \quad (13)$$

For a given symmetric positive definite Riemannian metric matrix g , we can decompose it that is, $g = h^T h$, where h and g are of the same order. Then,

$$ds^2 = dx^T g dx = (h dx)^T (h dx) = |h dx|^2. \quad (14)$$

In other words, the distance is converted into the module length of $h dx$. Then, the matrix h just describes the local coordinate system. The vector dx in this coordinate system h is equivalent to the vector $h dx$ in the local rectangular coordinate system. At this time, h becomes the Jacobian matrix under coordinate transformation.

5.4. Network Behavior Risk Calculation. Because there is no concept of measure in differential manifold in the process of using differential manifold to measure network security, it is necessary to give Riemannian measure g on differential manifold, so that the behavior risk value of network can be calculated through differential manifold. The Riemannian metric g selected in this study is as follows:

$$g_{uv} = \begin{cases} e^{(x_u^2 + x_v^2)}/2 & u = v, \\ 0 & u \neq v, \end{cases} \quad (15)$$

where x_u and x_v are the coordinates of point u and point v , respectively. After the Riemannian metric is determined, the corresponding geometric quantity can be given on it. For any vector $A = (a_1, a_2, \dots, a_n)$ in a Riemannian manifold, the module length of the vector can be expressed as follows:

$$\begin{aligned} |\hbar A| &= \sqrt{(\hbar A)^T (\hbar A)} = \sqrt{A^T \hbar^T \hbar A} \\ &= \sqrt{A^T g A} = \sqrt{\sum_{u=1}^n g_{uu} a_u^2} \end{aligned} \quad (16)$$

For any vector A and any vector B in a manifold, their inner product in a Riemannian manifold can be expressed as follows:

$$(\hbar A)^T (\hbar B) = A^T \hbar^T \hbar B = A^T g B. \quad (17)$$

It can be seen from Section 4.5 that the network behavior risk result can be expressed by the hypervolume of the high-dimensional data manifold formed by the network security index.

Formula (15) has given the selected Riemannian manifold. Combined with the differential manifold structure of the network system, it can be seen that in the Riemannian manifold, the volume element of the data manifold can be expressed as follows:

$$\begin{aligned} \det(\hbar) \prod_u dx^u &= \sqrt{\det(\hbar^T \hbar)} \prod_u dx^u \\ &= \sqrt{\det(g)} \prod_u dx^u \\ &= \sqrt{\det(g)} d\Omega, \end{aligned} \quad (18)$$

where g is the selected Riemannian metric in Riemannian manifold and $\sqrt{\det(g)}$ represents the volume scaling factor of Riemannian manifold space relative to Euclidean space. Given n vectors A^1, A^2, \dots, A^n in a manifold, the super volume composed of these vectors can be expressed as follows:

$$R = \int \Omega \sqrt{\det(g)} d\Omega. \quad (19)$$

According to formula (19), we can get the network behavior risk measurement value, which can be used to evaluate the network risk quantitatively and judge the security state of the network.

6. Experimental Design and Analysis

6.1. Small-Scale Network Environment Experiment. In order to verify the effectiveness of the measurement method, this study builds a network environment to simulate DoS attacks. First, we set up an experimental environment and then collect the experimental data. Finally, we use the proposed model method to calculate and draw the conclusion. In order to simulate the attack, four attackers are set up in the network system to simulate DoS attacks of different scales. The attack traffic enters the internal network through the router R1, as shown in Figure 5. This study uses LOIC attack to simulate DoS attack and Wireshark to collect data index.

The experimental design process based on real small environment is as follows: (1) Collection and pretreatment of indicators. Wireshark tool is used to collect data indicators and preprocess them. (2) Calculate the network benchmark security value, namely, network security baseline. (3) Calculate the network risk value under different DoS attack scales. (4) According to the calculated network benchmark

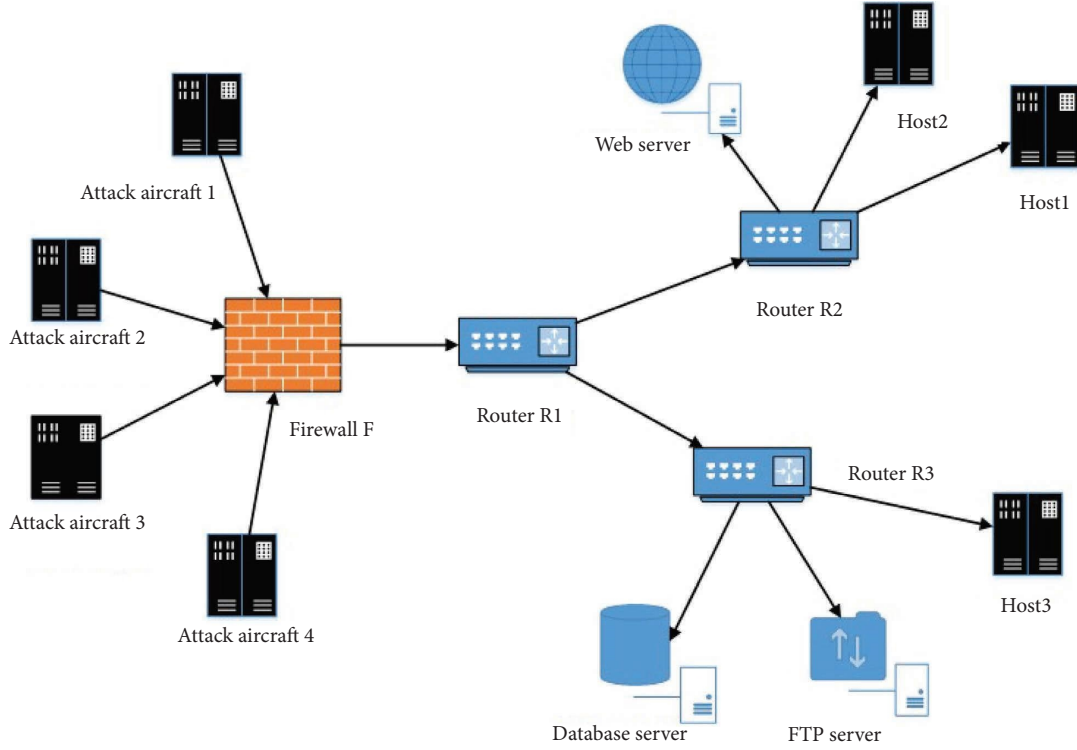


FIGURE 5: Network environment topology.

security value and network security risk value, the effectiveness of the measurement method is judged.

The calculation process and results are as follows:

- (1) Construct the network index state matrix
Collect the data in a certain period of time to form the network index state matrix.
- (2) Index pretreatment
For the collected data, the data are standardized and converted into dimensionless pure values between [0, 1].

$$S = \begin{bmatrix} 0.11 & 0.40 & 0 & 0.03 & 0.34 & 0.07 \\ 0.02 & 0.40 & 0.14 & 0 & 0.68 & 0 \\ 0.12 & 0.40 & 0.01 & 0 & 0.59 & 0 \\ 0.12 & 0.40 & 0.02 & 0.02 & 1 & 0.05 \\ 0.21 & 0.07 & 0.02 & 0.06 & 0.63 & 0.04 \\ 0.11 & 0.70 & 0.02 & 0 & 0.81 & 0 \end{bmatrix}. \quad (20)$$

- (3) Construct the network Riemannian metric

$$g_{uv} = \begin{cases} e^{(x_u^2 + x_v^2)/2} & u = v, \\ 0 & u \neq v. \end{cases} \quad (21)$$

In this case, the Riemannian metric matrix under the above network state matrix S' is expressed as follows:

$$G = \begin{bmatrix} 1.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.17 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.49 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (22)$$

- (4) Calculate the volume scaling factor

Firstly, the volume scaling factor of Riemannian manifold relative to Euclidean space is calculated. It can be seen from the above that if the currently selected Riemannian metric matrix is G , the determinant of Riemannian metric matrix can be obtained as $A = |G| = 1.76$. Therefore, the volume scaling factor of Riemannian manifold relative to Euclidean space is 1.33.

- (5) Calculate the network security baseline

In the Riemannian manifold of data matrix, the volume element can be expressed as follows:

$$\det(\hbar) \prod_u dx^u = \sqrt{\det(G)} d\Omega = 1.33d\Omega. \quad (23)$$

Then, the super volume of index data manifold is as follows:

$$R = \int \Omega \sqrt{\det(G)} d\Omega = 1.33 \int \Omega d\Omega = 2.905. \quad (24)$$

From the above, it can be seen that in the small network environment when the network is running normally, the network security benchmark result is 2.905. Using the same calculation method, we can calculate the network security risk value of DoS attack under different scales, as shown in Table 4.

6.2. CIC Open Data Set Experiment. In order to explain the accuracy of the network behavior measurement method based on differential manifold, this study analyzes the CIC-IDS-2017 public data set [36] and draws a conclusion by comparing the calculation results of the open data set on Monday and other time periods. CIC-IDS-2017 data set builds abstract behaviors of 25 users based on HTTP, FTP, and other protocols, including common attacks and traffic analysis results [36]. This study calculates the network behavior risk value based on the attack data from Tuesday to Friday, compares the network security benchmark value under normal conditions on Monday, and draws a conclusion.

The experimental design process based on CIC public data set is as follows: (1) Collection of indicators. Each group of data of the same data tag is collected to form a number of index matrix V_i . Then, the new index state matrix V'_i is obtained by preprocessing and dimensionality reduction. (2) Calculate network benchmark security value. The data set on Monday is the index data collected for normal operation, and the index state matrix of normal data after dimensionality reduction is set as V'_1 and the network benchmark security value is calculated. (3) Calculate the network security risk value under each attack. The dataset collected data from Tuesday to Friday, including DoS attack, Heartbleed attack, Web attack, and other types of attacks. Analyze the index state matrix $V'_i (i = 2, 3, \dots)$ of these attacks and calculate the corresponding network security risk values, respectively. (4) According to the calculated network benchmark security value and network security risk value, judge the network security state in the attack period and draw a conclusion.

The experimental results are as follows.

In CIC data set, Monday is normal data and the collection time is 8:55–10:27, a total of more than 500000 sets of data and a total of more than 80 indicators. After calculation, the network benchmark security value is 0.17. The experimental results are shown in Table 5.

6.3. Experimental Analysis and Conclusion. For the small physical network environment, the experimental verification shows that the network security benchmark value is 2.905 when the network is running normally. In addition, the network security risk value after DoS attack is greater than 2.905 and the risk multiple is about 3 times. It indicates that the current network is under external attack, and the network is in an insecure situation. This is exactly the same as the actual situation. Based on the experimental verification

TABLE 4: Network security risk calculation results.

Operation status	Security risk value
Normal operation	2.905
The first DoS attack	8.053
The second DoS attack	9.976

of small-scale network environment, the security status values of the network in the normal state, the first DoS attack, and the second DoS attack are compared, which shows the effectiveness of the model method proposed in this study.

As shown in Figure 6, the normal fluctuation range of network risk is set between (0, 0.32) that is, the network security risk baseline is set to 0.32. As can be seen from Figure 6, the network security state calculated by the network behavior risk measurement method based on differential manifold is basically consistent with the attack tag given in CIC-IDS-2017. By comparing the calculation results of network activity risk under normal conditions on Monday and from Tuesday to Friday, it can be seen that the network behavior risk measurement method based on differential manifold proposed in this study is basically effective and can detect most of the network attacks, which proves the effectiveness of the network behavior risk measurement method based on differential manifold.

Furthermore, we use differential manifold to illustrate the accuracy of behavior risk measurement (BRM). In Section 6.2, we used BRM and several traditional measurement methods based on machine learning to measure the risk of CIC open dataset. We use the following three common information retrieval evaluation indicators: precision (PR) is the proportion of the number of positive instances correctly classified to the number of instances classified as positive instances; recall (RC) is the proportion of the predicted number of all positive samples in the data set to all positive samples; F -measure ($F1$) is a weighted harmonic average of accuracy rate and recall rate. In addition, the execution time of the test process is calculated and displayed in Table 6. The result of the comparison between several mentioned machine learning methods in reference [37] and BRM is shown in Table 6. We can observe that the execution time of KNN is 1908.23 seconds which is the slowest, while that of BRM is 226 seconds. According to the weighted average of the three evaluation indexes PR, RC, and $F1$, the BRM algorithm has a high accuracy. In addition, these traditional machine learning measurement methods rely on a large amount of data training in the test process. By comparing the accuracy, recall rate, and execution time of the measurement methods, the performance of the proposed method is better than that of some measurement methods based on machine learning, that further proves the accuracy of the model method in this study.

In CIC-IDS-2017 dataset experiment, this study proposes a network behavior risk measurement method based on differential manifold which has limitations under SSH attack and BOT attack. Compared with other attacks, the fluctuation of network security risk under SSH

TABLE 5: Experimental result.

Time	Attack type	Network security risk value
Monday	Nothing	0.17
Tuesday	FTP attack	25.9
	SSH attack	0.297
Wednesday	DoS slowhttptest attack	4.15
	DoS slowloris attack	1.44
	DoS goldeye attack	0.63
	Heartbleed attack	72.9
Thursday	XSS attack	53
	Sql injection attack	98.3
	Brute force attack	53.7
	Infiltration attack	267
Friday	Bot attack	0
	DDoS attack	0.403
	Portscan attack	0.582

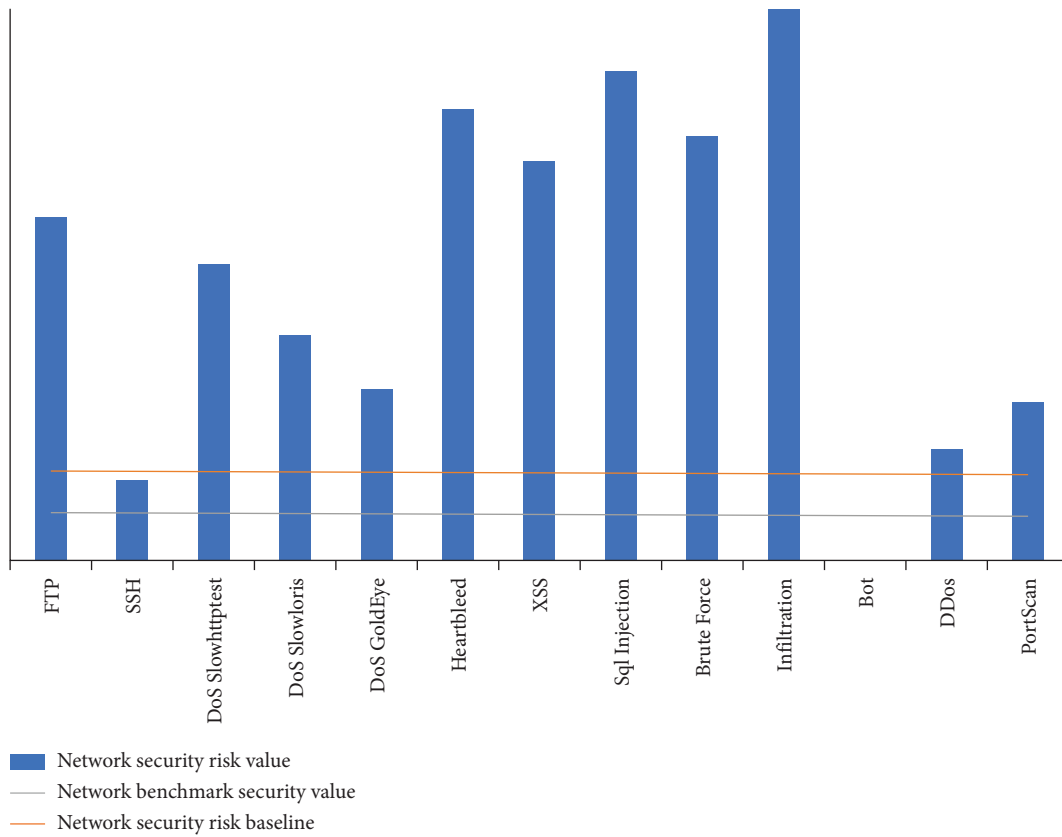


FIGURE 6: Comparison of network security results under various attacks.

TABLE 6: Comparison table of various network security measurement methods.

Common methods	Pr	Rc	F1	Execution time (s)
KNN	0.96	0.96	0.96	1908.23
RF	0.98	0.97	0.97	74.39
AdaBoost	0.77	0.84	0.77	1126.24
MLP	0.77	0.83	0.76	575.73
Naïve Bayes	0.88	0.04	0.04	14.77
QDA	0.97	0.88	0.92	18.79
ID3	0.98	0.98	0.98	235.02
BRM	0.85	0.85	0.85	226

attack is small and the calculated network risk value is also small. BOT attacks are malicious code that invades the network. BOT attacks on the network are difficult to detect, and the attack characteristics are not obvious enough.

7. Conclusion

This study measures the network behavior risk based on the traffic analysis, regards the network traffic data as the network behavior, depicts the network system as a differential manifold, determines the network risk measurement index group, collects the network operation index data for pre-processing, and uses the differential manifold theory to calculate the security benchmark value under the normal operation of the network and the network security risk value under the attack, and draw a conclusion by comparison. Finally, the proposed method is verified by comparative experiments.

The following three aspects are focused on this study: (1) Regard network traffic as network behavior and propose the definition and measurement method of network behavior risk. Through traffic analysis, carry out risk measurement and realize the quantitative analysis of network behavior. (2) Use the local linear embedding dimension reduction algorithm of manifold learning to reduce the dimension of the index. (3) Apply the differential manifold theory to network security measurement. With the help of differential manifold theory, the description of the security state of the whole network system can be transformed into the change of the state in the high-dimensional manifold composed of network indicators, and then the network security activities are abstracted in the high-dimensional space to calculate the network behavior risk value.

This study focuses on the method of measuring network behavior risk by differential manifold but still has some limitations including (1) When it comes to the measurement of the network security, less attention is paid to the measurement of other indicators such as the vulnerability of the network itself, while more attention is paid to the drastic changes of indicators in network attack and defense. The follow-up research needs to add the measurement of assets and network vulnerability on the basis of the existing network activity measurement. (2) Use the dimension reduction method of local linear embedding to reduce the dimension of data. The performance of the local linear embedding algorithm mainly depends on the selection of nearest neighbor number. A large number of nearest neighbors will cause the smoothness of manifold, and too few nearest neighbors may divide disjointed submanifolds. The subsequent research on dimension reduction parameters can optimize the measurement method and improve the measurement accuracy.

Data Availability

CIC-IDS-2017 public data set is provided by Canadian Institute of cyber security.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank for CIC-IDS-2017 public data set provided by Canadian Institute of cyber security. This work was supported by the National Key Research & Development Program of China (2020YFB1712104) and Major Scientific and Technological Innovation Projects of Shandong Province (2020CXGC010116).

References

- [1] G. Wang, H. Zhang, and D. Chen, "Risk analysis of network security," *Modern Computer*, vol. 8, pp. 46–48, 2001.
- [2] J. Wang, K. Fan, W. Mo, and D. Xu, "A method for information security risk assessment based on the dynamic bayesian network," in *Proceedings of the 2016 International Conference on Networking and Network Applications (NaNA)*, pp. 279–283, IEEE, Hakodate, Japan, June, 2016.
- [3] Y. Yang, "Ddos attack detection of internet of things based on traffic (in Chinese)," Master's Thesis, Beijing Jiaotong University, 2020.
- [4] L. Hu, H. Li, Z. Wei, S. Dong, and Z. Zhang, "Summary of research on it network and industrial control network security assessment," in *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1203–1210, IEEE, Chengdu, China, March, 2019.
- [5] X. Lei, T. Ma, Z. Niu, C. Ma, and H. Shan, "Research on ad hoc network security risk assessment method," in *Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, pp. 2272–2279, IEEE, Chongqing, China, June, 2020.
- [6] Y. Ye, L. Yan, W. Sun, Q. Zhang, and N. Wang, "Discussion on risk assessment of network security management," in *Proceedings of the 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pp. 409–411, IEEE, Changsha, China, February, 2018.
- [7] A. Ramos, M. Lazar, R. H. Filho, and J. Rodrigues, "Model-based quantitative network security metrics: a survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2704–2734, 2017.
- [8] L. Yu, Y. Sheng, and Z. Pan, "Hierarchical quantitative evaluation of vulnerability exploitability," in *Proceedings of the 2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, pp. 115–118, IEEE, Harbin, China, June, 2018.
- [9] J. David and C. Thomas, "Efficient ddos flood attack detection using dynamic thresholding on flow-based network traffic," *Computers and Security*, vol. 82, pp. 284–295, 2019.
- [10] Z. Liu, C. Hu, and C. Shan, "Riemannian manifold on stream data: Fourier transform and entropy-based ddos attacks detection method," *Computers and Security*, vol. 109, no. 10, Article ID 102392, 2021.
- [11] M. Di Penta and D. A. Tamburri, "Combining quantitative and qualitative studies in empirical software engineering research," in *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion*

- (ICSE-C), pp. 499-500, IEEE, Buenos Aires, Argentina, May, 2017.
- [12] M. Sheng, H. Liu, X. Yang, W. Wang, J. Huang, and B. Wang, "Network security situation prediction in software defined networking data plane," in *Proceedings of the 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pp. 475-479, IEEE, Dalian, China, August, 2020.
- [13] J. Wang, N. Zeng, and Z. Hu, "Research on information security risk assessment of computer network based on gray analytic hierarchy process," in *Proceedings of the 2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC)*, pp. 1429-1434, IEEE, Dalian, China, December, 2017.
- [14] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 Workshop on New Security Paradigms*, pp. 71-79, Charlottesville, VA, USA, September, 1998.
- [15] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61-74, 2012.
- [16] S. Zavrak and M. Iskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, no. 99, pp. 108346-108 358, 2020.
- [17] E. Biersack, C. Callegari, and M. Matijasevic, "Data traffic monitoring and analysis: from measurement, classification, and anomaly detection to quality of experience," *Lecture Notes in Computer Science*, vol. 5, no. 23, pp. 12561-12570, 2013.
- [18] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proceedings of the 2017 International Conference on Information Networking (ICOIN)*, pp. 712-717, IEEE, Jeju Island, Korea, January, 2017.
- [19] N. Marir, H. Wang, G. Feng, B. Li, and M. Jia, "Distributed abnormal behavior detection approach based on deep belief network and ensemble svm using spark," *IEEE Access*, vol. 6, pp. 59 657-659 671, 2018.
- [20] A. Shubair, S. Ramadass, and A. A. Altyeb, "kenfis: knn-based evolving neuro-fuzzy inference system for computer worms detection," *Journal of Intelligent and Fuzzy Systems*, vol. 26, no. 4, pp. 1893-1908, 2014.
- [21] X. Zhao, Y. Zhang, J. Xue, C. Shan, and Z. Liu, "Research on network risk evaluation method based on a differential manifold," *IEEE Access*, vol. 8, pp. 66 315-366 326, 2020.
- [22] C. Hu, "Calculation of the behavior utility of a network system: conception and principle," *Engineering*, vol. 4, no. 1, pp. 78-84, 2018.
- [23] Z. Hao, *Research on nonlinear dimensionality reduction method based on differential manifold*, Shanghai University, Ph.D. dissertation, 2016.
- [24] X. Mei and L. He, *Differential Manifolds and Riemannian Geometry*, (in chinese), Differential Manifolds and Riemannian Geometry, Gulf Professional Publishing, Houston, TX, USA, 1987.
- [25] L. Wang, "Application of discriminative manifold learning algorithm in face recognition (in Chinese)," Master's Thesis, Chongqing University, 2009.
- [26] B. C. Hall, *Lie Groups, Lie Algebras, and Representations*, Springer Science and Business Media, Berlin, Germany, 2013.
- [27] X. Zhao, X. Jiang, J. Zhao, H. Xu, and J. Guo, "Measurement method of network attack and defense effectiveness based on differential manifold," *Journal of Tsinghua University: Natural Science Edition*, vol. 60, no. 5, p. 6, 2020.
- [28] C. Hu, Z. Liu, C. Shan, X. Zhao, and S. Guo, *Construction Method of Network State Model and State Evaluation Method Based on Differential Manifold*, 2018.
- [29] S. Zhao, C. Wu, W. Xie, Z. Jia, H. Wang, and Y. Zhang, "Research on network security measurement based on attack graph," *Journal of Information Security*, vol. 4, no. 1, pp. 53-67, 2019.
- [30] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski, "Modeling internet backbone traffic at the flow level," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2111-2124, 2003.
- [31] W. Chen, *Preliminary of Differential Manifold*, Higher Education Press, Beijing, China, 1998.
- [32] Z. Bai, "Preliminary of riemannian geometry," in *Geometric Analysis of Quasilinear Inequalities on Complete Manifolds*, Springer Science and Business Media, Berlin, Germany, 2nd edition, 2004.
- [33] L. Hayden, *It Security Metrics a Practical Framework for Measuring Security and Protecting Data*, McGraw-Hill Education Group, New York, NY, USA, 2010.
- [34] R. Henning, M. Abrams, J. Kahn et al., "Information system security attribute quantification or ordering," in *Workshop on Information Security System Scoring and Ranking*, pp. 1-70, 2002.
- [35] L. Lin and C. Liu, "Research on key technologies of internet risk assessmen," *t Network Security Technology and Application*, vol. 4, p. 14, 2017.
- [36] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, *Intrusion Detection Evaluation Dataset (Cic-ids2017)*, Proceedings of the of Canadian Institute for Cybersecurity, Fredericton, New Brunswick, 2018.
- [37] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the International Conference on Information Systems Security and Privacy*, vol. 1, pp. 108-116, Copenhagen, Denmark, February, 2018.

Research Article

A Hybrid Approach by CEEMDAN-Improved PSO-LSTM Model for Network Traffic Prediction

Bilin Shao ¹, Dan Song ¹, Genqing Bian ², and Yu Zhao ¹

¹School of Management, Xi'an University of Architecture and Technology, Xi'an, China

²School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an, China

Correspondence should be addressed to Dan Song; 674355101@qq.com

Received 11 December 2021; Revised 23 June 2022; Accepted 15 July 2022; Published 12 September 2022

Academic Editor: Jinwei Wang

Copyright © 2022 Bilin Shao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As an important part of data management, network traffic evaluation and prediction can not only find network anomalies but also judge the future trends of the network. To predict network traffic more accurately, a novel hybrid model, integrating Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) with long short-term memory neural network (LSTM) optimized by the improved particle swarm optimization (IPSO) algorithm, is established for network traffic prediction. Firstly, an LSTM prediction model for the real-time mutation and dependence of network traffic is constructed, and the IPSO is applied to optimize the hyperparameters. Then, CEEMDAN is introduced to decompose sequences of raw network traffic data into several different modal components containing different information to reduce the complexity of the network traffic sequence. Finally, the evaluation of the experiments shows the feasibility and effectiveness of the proposed method by comparing it with other deep neural architectures and regression models. The results show that the proposed model CEEMDAN-IPSO-LSTM produced a significantly superior performance with a reduction of the prediction error.

1. Introduction

As the information storage terminals of the Internet of Things (IoT), and with the continuous expansion of the data center scale, the network structure of the data center is becoming increasingly complex, the network business and network data flow are growing rapidly, and the frequency of network congestion is also getting higher and higher [1–3]. Network traffic monitoring, network resource optimization, network congestion avoidance, and network security strategy are of great significance in the real-time analysis of network traffic [4–8]. When the network is overloaded or congested, accurate prediction can ensure high-quality execution of network services with super importance or priority [9]. In recent years, predictive analysis based on historical network traffic has become a major research topic in the academic field. Establishing an accurate prediction model to describe network traffic characteristics contributes to optimizing network topology structure and route planning, reducing

energy consumption, and providing more reliable service quality assurance.

Network links change dynamically with limited node processing resources. Network traffic prediction mainly depends on the statistical characteristics of flow and the strong correlation between time-sequence values. Modeling analysis based on network traffic time series is an effective method for network traffic research, which has been widely used in network traffic prediction and network performance evaluation. How to fully consider the complex characteristics of network traffic, and improve the prediction accuracy and real-time of network traffic has always been a hot and difficult topic of network traffic research [10].

In this study, we propose a novel hybrid model based on the Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN), the Long Short-Term Memory (LSTM) neural network model, and the improved particle swarm optimization (IPSO) algorithm to predict and analyze the network traffic. Firstly, combined with the real-time mutability, dependence, and highly nonlinear

characteristics of network traffic, we establish the LSTM network traffic prediction model to extract the dynamic characteristics of network traffic. Then, the IPSO is utilized for hyperparameters optimization. In addition, the CEEMDAN method is employed to decompose the network traffic data into several simplified modes. Finally, we compare the prediction accuracy of different models to evaluate the prediction effects of the CEEMDAN-IPSO-LSTM neural network model.

The main contributions of our work are presented as follows:

- (1) CEEMDAN is introduced to decompose network traffic data into several components, which creates modal confusion and avoids making larger impacts on the original signal during adding the white noise.
- (2) Network traffic prediction model based on LSTM is constructed to pursue the real-time mutation of network traffic.
- (3) The improved PSO algorithm is proposed to optimize the hyperparameters of the LSTM network traffic model. The optimization of hyperparameters of the LSTM prediction model can improve the prediction performance.

The rest of this paper is as follows. In section 2, we review the related research about network traffic prediction. In section 3, we constructed a network traffic prediction model based on LSTM. In section 4, we study the hyperparameter optimization of LSTM by the Improved Particle Swarm Optimization, and network traffic data denoising by CEEMDAN. Section 5 presents our evaluation of the proposed method. Finally, the main conclusions and future work are drawn in section 6.

2. Related Work

Due to the importance of network traffic prediction, there has been much research on network traffic prediction methods in recent years. Generally, network traffic prediction can be divided into short-term prediction, medium-term prediction, and long-term prediction according to the different cycles of network traffic prediction, [11] while network traffic prediction models are mainly divided into two categories: parametric model and nonparametric model [12].

2.1. Parametric model. The parametric model has the advantage of being simple and easy to understand. Moreover, it does not have high standards for training data, and the solving process is easy compared to nonparametric model, which consumes less time. However, the parametric model is suitable for the prediction of small data volumes with obvious features and stable structure, while the network traffic has real-time mutability and dependency characteristics, and the parametric model will lead to higher prediction errors than the nonparametric model.

ARMA network traffic model can effectively analyze network data with a stable flow in a short period, obtain network traffic characteristics at the corresponding scale,

and realize data flow decomposition [13]. In the network traffic model based on ARMA, the deployment of the multi-scale fitting process can obtain high accuracy under any expiration delay, simplify the ARMA model, and enhance the integration effect of the ARMA framework in network traffic modeling [14].

However, ARMA is not suitable for long-term network traffic data with network anomalies because the premise of ARMA modeling is that the data analyzed is a stationary random process. Most of the actual network traffic data are nonstationary [15], which can be transformed into stationary data after finite-difference. Therefore, some scholars proposed the Autoregressive Integrated Moving Average (ARIMA) model in the research process [16].

2.2. Nonparametric model. Nonparametric model refers to a model with no fixed structure and fixed parameters. Common nonparametric models include Support Vector Machine (SVM), k-Nearest Neighbor (kNN) [8], Artificial Neural Network (ANN), etc. The nonparametric model can automatically fit a variety of function forms without assumption, and the training effect is good, which is suitable for predicting large data volume.

Due to the real-time variability and dependence of network traffic, traditional network traffic prediction models have some disadvantages such as weak generalization ability and limited prediction accuracy. Therefore, more and more researchers use nonparametric models to predict network traffic data. The Support Vector Regression model (SVR) and its variant MK-SVR are first used to predict network traffic [17–19], which effectively predicts the changing trend of network traffic data but lacks the consideration of temporal correlation of time series data leading to a limit of prediction accuracy.

Methods based on the artificial neural network, such as Convolutional Neural Network (CNN) [20], improve the effect of flow classification by autonomous feature learning of data [21]. LSTM neural network and Gated Recurrent Unit (GRU) neural network have a superior effect over existing SVM and ANN models in predicting network traffic, which is more suitable for random nonlinear network traffic prediction [12]. LSTM neural network was originally used for short-term flow prediction, which can better learn the abstract representation of nonlinear flow data and capture the inherent characteristics of long-term dependence relationship in continuous data, thus improving the accuracy of flow prediction [22]. LSTM neural network is used for network traffic prediction, and the auto-correlation coefficient is added to the model to describe the trend of network traffic change better, which improves the accuracy of the prediction model [23]. On this basis, the improved Particle Filter (PF) algorithm is used to optimize the LSTM model, which improves the training rate and overcomes the shortcoming of convergence to local optimal in the traditional LSTM network [24].

The experiments of many neural network methods to predict the network traffic data show that in a real-time network data set, LSTM is of better performance than

Recurrent Neural Network (RNN), the Feed-forward Neural Network (FFN), and other classic methods. LSTM neural network can more accurately simulate time series and its long-term dependencies than the traditional RNN, in large network traffic matrix prediction, and obtain a faster convergence rate [25]. The variants of LSTM neural network, GRU neural network, and identity-RNN (IRNN) have comparable performance with LSTM [26]. Minimal Gated Unit (MGU) overcomes the shortcoming of the high computing cost of the LSTM network and achieves relatively predictable performance with less model training time [27]. In addition, LSTM neural network has achieved good prediction results in financial data forecast [28, 29], metal price prediction [30], air quality index prediction [31], modular temperature prediction [32], and bridge health monitoring [33].

In summary, a single parametric or nonparametric model has its problems and defects, while a hybrid prediction model can overcome the shortcomings of a single model by combining two or more models. The hybrid model mainly combines some decomposition algorithms, optimization algorithms, and prediction algorithms, respectively, in the data preprocessing, prediction, and result correction stage of network traffic prediction. Although combinatorial prediction has achieved good results in other researches [34, 35], there are still some problems, such as how to choose the prediction model and its parameters, how to integrate the prediction results reasonably, and how to choose the appropriate decomposition algorithm or optimization algorithm. For network traffic prediction, using the combined prediction model and overcoming the above problems is a research direction worthy of further study.

3. Network Traffic Prediction Based on LSTM

3.1. LSTM Neural Network Model. LSTM neural network (hereinafter referred to as LSTM) is an improvement of the recurrent neural network, which aims to overcome the defects of the recurrent neural network in processing long-term memory [36]. The LSTM introduced the concept of cellular states, which determine which states should be preserved and which should be forgotten. The basic principle of LSTM is shown in Figure 1.

As shown in Figure 1, X_t is the input at time t , h_{t-1} is the output of the hidden layer at time $t-1$, and C_{t-1} is the output of the historical information at time $t-1$; f , i , and, o are, respectively, the forgetting gate, input gate, and output gate at time t , and g is the internal hidden state, namely, the transformed new information. LSTM conducts parameter learning for them in the training. C_t is the updated historical information at time t , and h_t is the output of the hidden layer at time t .

Firstly, the input x_t at time t and the output h_{t-1} of the hidden layer are copied into four copies, and different weights are randomly initialized for them, to calculate the forgetting gate f , input gate i , and output gate o , as well as the internal hidden state g . Their calculation methods are shown in formulas (1)–(4), where W is the parameter matrix from the input layer to the hidden layer, U is the self-recurrent

parameter matrix from the hidden layer to the hidden layer, b is the bias parameter matrix, and σ is the sigmoid function, so that the output of the three gates remains between 0 and 1.

$$f = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad (1)$$

$$i = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad (2)$$

$$o = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad (3)$$

$$g = \sigma(W_g x_t + U_g h_{t-1} + b_g). \quad (4)$$

Secondly, forgetting gate f and input gate i are used to control how much historical information C_{t-1} is forgotten and how much new information g is saved, to update the internal memory cell state C_t . The calculation method is shown in formula (5).

$$C_t = f_t \otimes C_{t-1} \oplus i \otimes g. \quad (5)$$

Finally, output gate o is used to control how much C_t information of the internal memory unit is output to the implicit state h_t , and its calculation method is shown in formula (6).

$$h_t = o \otimes \tanh(C_t). \quad (6)$$

3.2. Network Traffic Prediction Model Based on LSTM. Network traffic data are modeled as a nonnegative matrix X of an $N \times T$, where N represents the number of nodes, T represents the number of time slots sampled, and each column in the data matrix represents the network traffic value at different nodes in a specific time interval.

Network traffic prediction can obtain the predicted value of the future time through the historical time series, $X(i, j)$ represents the scale of the $N \times T$ flow matrix, and $x_{n,t}$ represents the network traffic value of row n and column t . Network traffic prediction is defined by a series of historical network traffic data ($x_{n,t-1}, x_{n,t-2}, x_{n,t-3}, \dots, x_{n,t-1}$) to predict the network traffic at time t in the future. In the network traffic prediction model based on LSTM (Figure 2), it is assumed that the network traffic at a certain point in the t -slot is predicted, the input of the model is ($x_{n,t-1}, x_{n,t-2}, x_{n,t-3}, \dots, x_{n,t-1}$), and the output is the predicted value \hat{x}_t of the network traffic at the t -slot at this point.

In Figure 2, we summarize the process of network traffic prediction based on LSTM, and it mainly includes network traffic data preparation, data preprocessing (data resampling and null filling), normalization of data, data classification, prediction network building, network compilation, network evaluation, data prediction, and evaluation.

The detailed contents of each process for network traffic prediction are as follows:

- (1) Network traffic data preparation and preprocessing. To meet the time and frequency requirements (second, minute, hour, day, etc.) of network traffic data prediction, the original data are required to resample, namely, the time series from one

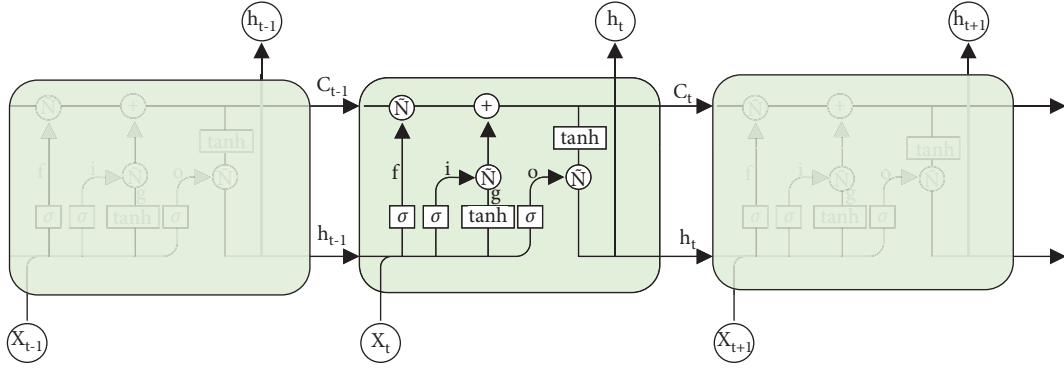


FIGURE 1: Basic principle of LSTM.

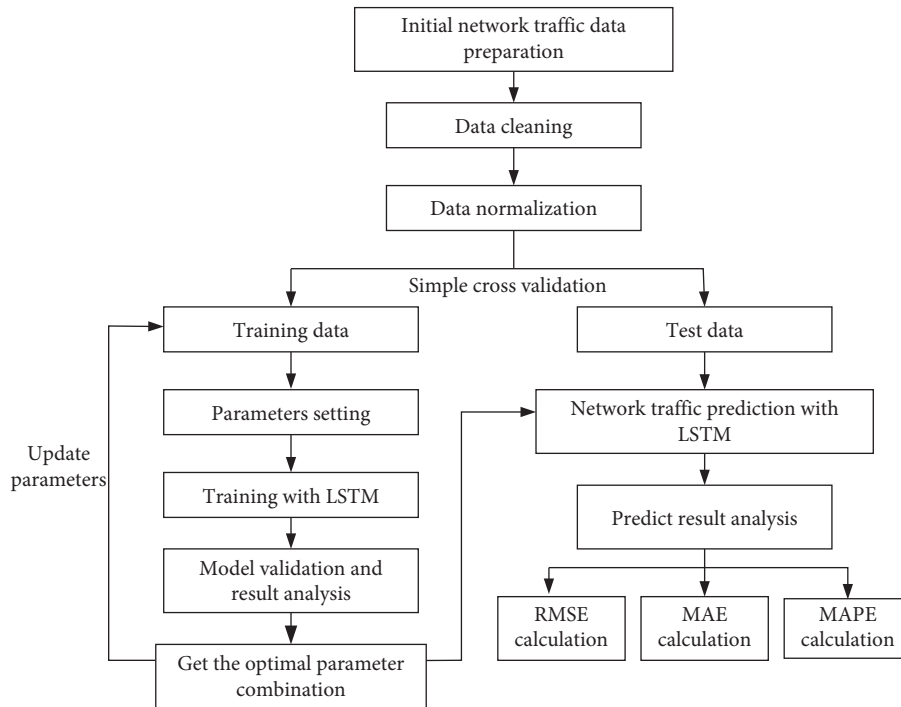


FIGURE 2: Process of LSTM network traffic prediction.

frequency is converted to another frequency. And to ensure even data time interval, the uneven time interval data are converted to equal interval data. There are generally two methods of data resampling: downward sampling and upward sampling. The former is to convert high-frequency data into low-frequency data, while the latter is to convert low-frequency data into high-frequency data. In addition, if there is a void value in the resampled data sequence, it is necessary to fill the void value. The commonly used methods include the direct deletion method, statistically based filling method, and machine-learning-based filling method. The direct deletion method may discard some important information in the data, and the statistically based filling method ignores the timing information of the data [37]. Therefore, this paper adopts the machine-

learning-based filling method—K-Nearest Neighbor (KNN) to fill the void value of network traffic data.

- (2) Data normalization. The range standardization method is used to process the network traffic data so that the sample data value is between 0 and 1. The calculation method of the range standardization method is shown in formula (7).

$$X_{\text{nor}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}, \quad (7)$$

where X_{max} represents the maximum value of network traffic data and X_{min} represents the minimum value of network traffic data.

- (3) Data division. After preprocessing and normalization, the network traffic data are divided into training set and test set according to the simple cross-

validation. Under the condition of keeping the network traffic data time sequence constant, the training set and the test set are divided by fivefold cross-validation [38], which are used for the training and prediction of the LSTM network traffic prediction model.

- (4) LSTM network traffic prediction model construction. An LSTM neural network is defined and its parameters are set, including the values of time step size, number of network layers, number of neurons in each layer, dropout, activation function, type and number of the return value, dimension size of the hidden layer, learning rate, batch processing size, iteration times, etc.
- (5) Network compilation. Set the optimizer, error measurement index, training record parameters, and compile the LSTM network traffic prediction model.
- (6) Network evaluation. Substitute training data into the model for training, and evaluate the error of the established prediction model. According to the results, finetune the parameter setting of the model to get a better prediction effect.
- (7) Prediction and evaluation. The optimized network traffic prediction model is used to make a prediction, and calculate the prediction errors by comparing prediction results with the real data.

4. The LSTM Network Traffic Prediction Model Optimized by IPSO and CEEMDAN

4.1. Improved Particle Swarm Optimization. Particle Swarm Optimization (PSO) is a simple-rule, fast-convergence-speed swarm intelligence optimization algorithm [39, 40]. It regards every individual as a part with no size and no quality in an n -dimension search space, which flies at a certain speed. It improves the searching through group cooperation and competition among the particles under the guidance of swarm intelligence.

Particle swarm optimization in n -dimensional continuous search space, for i -th ($i = 1, 2, \dots, m$) particle, determines that n dimensional current position vector $x^i(k) = [x^i_1, x^i_2, \dots, x^i_n]^T$ represents the current position of the i -th particle in the search space, and n dimensional velocity vector $v^i(k) = [v^i_1, v^i_2, \dots, v^i_n]^T$ represents the search direction of the particle. The optimal position (pbest) experienced by the i -th particle in the group is denoted as $p^i(k) = [p^i_1, p^i_2, \dots, p^i_n]^T$, and the optimal position (gbest) experienced by all particles in the group is denoted as $p^g(k) = [p^g_1, p^g_2, \dots, p^g_n]^T$. The basic PSO algorithm is shown in formulas (8) and (9).

$$v^i_j(k+1) = \omega(k)v^i_j(k) + c_1 \text{rand}(0, a_1)(p^i_j(k) - x^i_j(k)) + c_2 \text{rand}(0, a_2)(p^g_j(k) - x^i_j(k)), \quad (8)$$

$$x^i_j(k+1) = x^i_j(k) + v^i_j(k+1), \quad (9)$$

where $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$, $\omega(k)$ is the inertia weight factor, c_1 and c_2 is acceleration constant, all of which are nonnegative values. $\text{rand}(0, a_1)$ and $\text{rand}(0, a_2)$ are random numbers with uniform distribution within the range of $[0, a_1]$ and $[0, a_2]$, a_1 and a_2 are corresponding control parameters.

In the PSO algorithm, ω keeps the particle moving inertia so that it tends to expand the search space, the ability to search new areas. The ω value usually adopts the linear inertia weight method, that is, the ω value increases or decreases linearly with the number of iterations. Compared with the fixed ω value, the linear method improves the optimization ability and convergence speed of the PSO algorithm to some extent, but it is far from enough. The nonlinear inertia weight method can further improve the optimization ability and convergence speed of the PSO algorithm [41]. Therefore, the ω calculation in this paper is improved by using the nonlinear inertia weight method, as shown in formula (10).

$$\omega = \omega_{\max} - (\omega_{\max} - \omega_{\min}) * \arcsin \frac{i}{\text{item_max}} * \frac{2}{\pi} \quad (10)$$

In formula (10), ω_{\max} and ω_{\min} , respectively, represent the maximum inertia weight and the minimum inertia weight, and i is the current iteration number. item_max is the maximum iteration number.

In the PSO algorithm, c_1 and c_2 are used to adjust the step size of particle movement. In this paper, the sine function is used to improve the acceleration constant [29]. The calculation method is shown in formulas (11) and (12).

$$c_1 = 2 \sqrt{1 - \sin\left(\frac{\pi}{2} * \frac{i}{\text{item_max}}\right)}, \quad (11)$$

$$c_2 = 2 \sqrt{\sin\left(\frac{\pi}{2} * \frac{i}{\text{item_max}}\right)}. \quad (12)$$

4.2. LSTM Hyperparameter Optimization Based on Improved PSO. The selection of hyperparameters of the LSTM prediction model has an important influence on prediction accuracy. The current hyperparameter selection method based on the empirical method has randomness, blindness, and nonuniversality in the parameter setting. Therefore, multiple hyperparameters are formed into a multidimensional solution space, and the optimal parameter combination is obtained by traversing the solution space, which can reduce the randomness and blindness of parameter selection. Multiple hyperparameter selections are in a larger scope, which needs a better performance optimization algorithm to obtain the global optimal solution quickly, so we introduce the improved particle swarm algorithm (Improved PSO, IPSO) to optimize LSTM model parameters. With the quick convergence speed, the IPSO promotes the scientific nature of the model parameter selection and further improves the prediction accuracy of the models.

It is assumed that n hyperparameters of the LSTM network traffic prediction model are optimized, each particle represents a set of hyperparameters of solution space. It is

supposed in the n -dimensional continuous search space, there are m groups of hyperparameter combinations, representing the i -th ($i = 1, 2, \dots, m$) hyperparameter. The current position vector $x^i(k) = [x_1^i x_2^i \dots x_n^i]^T$ of n dimension represents the current value of an i -th group of hyperparameters in the solution space. The velocity vector $v^i(k) = [v_1^i v_2^i \dots v_n^i]^T$ of n dimension represents the search direction of this group of hyperparameters.

The goal of network traffic prediction is to make the predicted value close to the actual value, that is, the error between the predicted value and the actual value is as small as possible. Therefore, the Root Mean Square Error (RMSE) of training data in the network traffic prediction model is selected as the objective function. Let $\text{fitness} = \text{RMSE}$, then the objective function is to minimize RMSE. The RMSE calculation method is shown in formula (13).

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}. \quad (13)$$

In formula (13), \hat{y}_i is the prediction value. $\hat{y} = \{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_i\}$, y is the real value, $y = \{y_1, y_2, \dots, y_i\}$.

Two important hyperparameters of the LSTM network traffic prediction model are optimized according to IPSO: time step size and the number of neurons in each layer. The single-layer and bilayer LSTM models are taken as the research objects to carry out the hyperparameter optimization. For the single-layer LSTM model, the node is for the number of neurons, and the lookback is for the time step, $\text{fitness} = \text{RMSE}(\text{node}, \text{lookback})$; for a bilayer LSTM model, $\text{fitness} = \text{RMSE}(\text{node1}, \text{node2}, \text{lookback})$.

According to the algorithm flow of IPSO, the process of IPSO optimized LSTM network traffic prediction model hyperparameter mainly includes six steps.

Step 1. The IPSO parameter is set. The particle swarm size is set as the number of hyperparameter combinations m . Each particle is randomly set as the initial value and speed of each group of hyperparameters within the allowed range. The maximum number of iterations item_max and the prediction error Pre_error .

Step 2. The fitness of each particle is evaluated, that is, the fitness value of the objective function of each group of hyperparameters is calculated.

Step 3. The optimal objective function value P_i for each set of hyperparameters is set. For the i -th group hyperparameter, its current target function value current_fitness is compared with P_i . If it is less than P_i , then current_fitness is used as the best target function value P_i for the i th group hyperparameter, namely, $P_i = \text{current_fitness}$.

Step 4. The global optimal value P_g . For the hyperparameter of i -th group, P is compared with P_g . If it is less than P_g , then P_i is taken as the optimal value P_g of the current group, namely, $P_g = P_i$.

Step 5. The search direction and value of each set of hyperparameters are updated according to formulas (8) and (9).

Step 6. The termination conditions are checked. If the set condition (default error or the maximum number of iterations) is not met, step 2 is returned to continue execution.

4.3. Network Traffic Data Decomposition by CEEMDAN. The empirical mode decomposition algorithm (EMD) is a data processing method commonly used for nonstationary time series signals [42]. It can decompose the nonstationary signals into a series of intrinsic mode function (IMF) components with different time scales. However, modal confusion exists in this method. Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) algorithm improved the EMD algorithm by adding a set of white noise with equal size and opposite signs before decomposing data via the EMD [43]. The CEEMDAN both confuses modal confusion and also avoids making larger impacts on the original signal during adding the white noise. The main steps of CEEMDAN are as follows:

- (1) Add a group of Gaussian white noise sequence $\varepsilon_i(t)$ with opposite signs to the original sequence $x(t)$, and obtain a new set of time series;

$$\begin{cases} x_i^+(t) = x(t) + \varepsilon_i^+(t), \\ x_i^-(t) = x(t) + \varepsilon_i^-(t). \end{cases} \quad (14)$$

- (2) Decompose each time series via EMD in formula (15) and obtain n intrinsic mode functions components;

$$\begin{cases} x_i^+(t) = \sum_{j=1}^n c_{ij}^+(t), \\ x_i^-(t) = \sum_{j=1}^n c_{ij}^-(t), \end{cases} \quad (15)$$

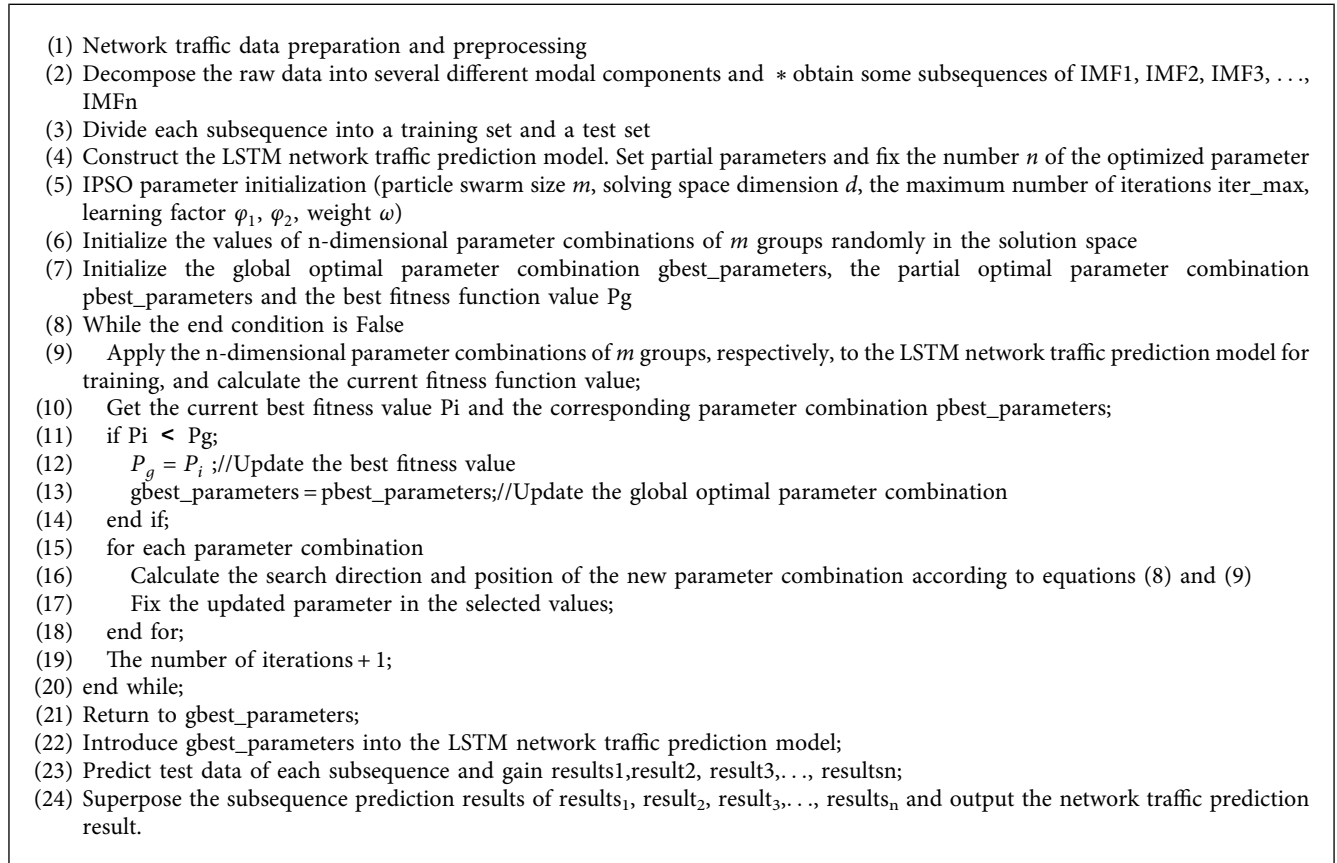
where, c_{ij} is the j -th modal component obtained by EMD decomposition after adding white noise for the i -th time.

- (3) Add different adaptive noises and repeat steps (14) and (15) for m times to obtain the set of m groups of intrinsic modal components (IMF), in which the last group is the trend term (Res);
- (4) Calculate the ensemble average of all components to obtain the final modal component group $c_i(t)$.

$$c_i(t) = \frac{1}{2m} \sum_{j=1}^m (c_{ij}^+(t) + c_{ij}^-(t)). \quad (16)$$

The process of network traffic prediction based on IPSO-LSTM combined with CEEMDAN is shown in Figure 3.

The process of data decomposition and prediction includes three main steps.



ALGORITHM 1: CEEMDAN-IPSO-LSTM network traffic prediction algorithm.

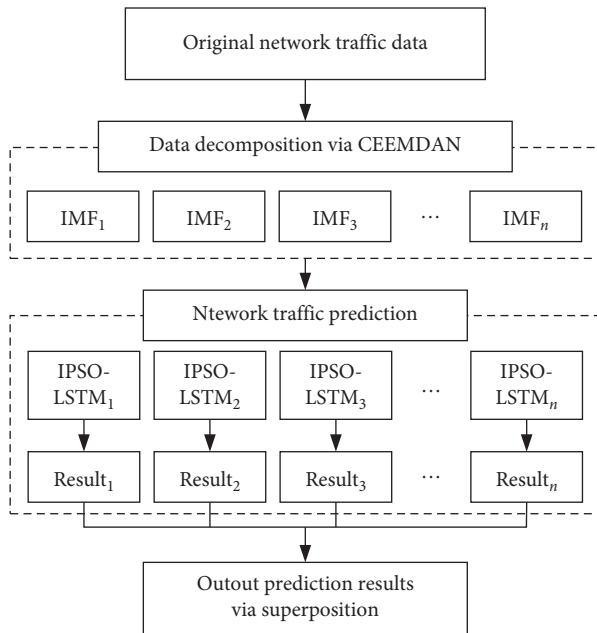


FIGURE 3: Process of data decomposition and prediction.

- (1) The network traffic data are decomposed by CEEMDAN into several different modal components and obtain some subsequences of IMF₁, IMF₂, IMF₃, ..., IMF_n;

- (2) Use the IPSO-LSTM model to predict each subsequence and gain results₁, result₂, result₃, ..., results_n;
- (3) Superpose the subsequence prediction results of results₁, result₂, result₃, ..., results_n and output the network traffic prediction result.

4.4. Network Traffic Prediction Algorithm Based on CEEMDAN-IPSO-LSTM. According to the process of IPSO for hyperparameter optimization and data de-composition by CEEMDAN, based on the network traffic prediction steps of LSTM, the CEEMDAN-IPSO-LSTM network traffic prediction algorithm is obtained. The pseudo-code of the algorithm is shown in Algorithm 1.

Algorithm 1 firstly prepares network traffic data and decomposes the raw data into several subsequences, and then divides each subsequence into a training set and a test set. Then, it uses the IPSO-LSTM network traffic model to obtain the optimal parameter combination. Finally, the optimal parameters are substituted into the LSTM model to complete the prediction of each subsequence and output the network traffic prediction result by superposing subsequence prediction results.

CEEMDAN-IPSO-LSTM network traffic prediction algorithm contains three processes, the time complexity of data decomposition is $O(k^2)$, k is the size of the predicted data set; the time complexity of hyperparameter optimization process is $O(n!)$; and the time complexity of the

prediction process is $O(hp + h^2 + h)$, h is the hidden_size, p is the input_size. Therefore, the time complexity of CEEMDAN-IPSO-LSTM is

$$T(n) = O(k^2) + O(n!) + O(hp + h^2 + h). \quad (17)$$

In the running process of the algorithm, the parameter optimization process consumes the most time with the highest computational complexity, but its time cost is acceptable because this process needs to run only once to obtain the optimal combination of hyperparameters. Once the hyperparameters are determined, the main time complexity is reflected in the prediction process. The time of the prediction process is mainly spent in the training. As long as the training is completed, the prediction can be finished by substituting the input data into the equation.

5. Experiment Evaluation and Discussion

5.1. Experimental Environment Configuration and Parameter Setting. This experiment completed under the measured flow data of BC-Oct89Ext provided by Bell Laboratory is selected. The flow data were Ethernet data detected in The Bell Morristown study, containing one million packets. This paper selects some data segments of BC-Oct89Ext flow data for model analysis.

For the prediction results of the network model, three error analysis indicators were used to verify the prediction accuracy, which were Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE), respectively. MAE and MAPE calculation methods are shown in equations (18) and (19).

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|, \quad (18)$$

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100\%. \quad (19)$$

According to Equation (13), the smaller the RMSE value, the smaller the average error between the prediction results and the actual data, the higher the prediction accuracy of the model, and the better the prediction performance of the model. Similarly, it can be seen from equations (18) and (19) that the MAE and MAPE values tend to 0, the better the prediction effect of the model is and the more perfect the model is. On the contrary, the greater the value is, the greater the error is, and the worse the prediction effect of the model is.

5.2. Network Traffic Prediction Results Based on LSTM

5.2.1. Data Processing

(1) Data resampling. As the original network traffic data in BC-OCT89Ext were collected multiple times per second with unequal time intervals, the data collected multiple times per second were preprocessed with the mean value method, and then the K-Nearest Neighbor (KNN) algorithm was used to fill the void value. Figure 4 shows 1800 pieces of flow data after packet resampling and null value processing.

(2) Data decomposition. It can be seen from Figure 5 that network traffic data have obvious nonlinearity and non-stationarity, which makes prediction difficult. Then the original time series is decomposed by the CEEMDAN method into several more predictable time subseries, and six groups of modal subsequences were obtained from high frequency to low frequency. Decomposition results are shown in Figure 5. It can be seen that the fluctuation of IMF1 to Res subsequence gradually flattens out and the frequency becomes lower and lower.

(3) Data division. The data after normalization was divided into a training set and a test set according to simple cross-validation. The first 80% of the data were used as training data for LSTM network model training. The remaining 20% of the data were used as prediction data to verify the efficiency of the model.

5.2.2. Network Traffic Prediction Based on Basic LSTM

(1) Network definition. In this forecast, the network structures of three-layer LSTM (one input layer, one hidden layer, and one output layer) and four-layer LSTM (one input layer, two hidden layers, and one output layer) are, respectively, adopted.

The specific connection mode of the three-layer LSTM is as follows: the timesteps of LSTM in the first layer are 1. The input of the data dimension is 3, and the number of neurons is 64. The second layer hidden layer (dense) takes the output of the first layer LSTM as input; the output layer of the third layer takes the output of the second hidden layer as the input and connects to a full connection layer. A one-dimensional vector with a length of 360 output from the full connection layer is the final output result, which represents the value of the predicted future 360 data points. To prevent overfitting, a dropout layer was added between the first layer and the hidden layer for regularization. After many tests in this experiment, it was concluded that when the dropout is 0.3, the training set had the highest accuracy.

Compared with the three-layer LSTM network, a hidden layer is added to the four-layer LSTM network structure. The hidden layer uses the results of the first layer as the input for training and transmits its output to the next hidden layer. The number of neurons is the same as that of the first layer. The dropout = 0.3 layer is added in both the first and second layers to prevent overfitting.

(2) Network compilation. LSTM network compilation uses the adaptive moment estimation (Adam) algorithm as the optimizer and the mean square error loss function as the objective function.

(3) Network fitness. The LSTM network was trained on 1440 samples and 360 samples were used for testing. The number of iterations epochs equals 50, look_back is made of 1, 5, and 10, respectively, and batch_size equals 128.

(4) Network evaluation. When look_back takes 1, 5, and 10, respectively, and the number of hidden layers (LN) is 1 and

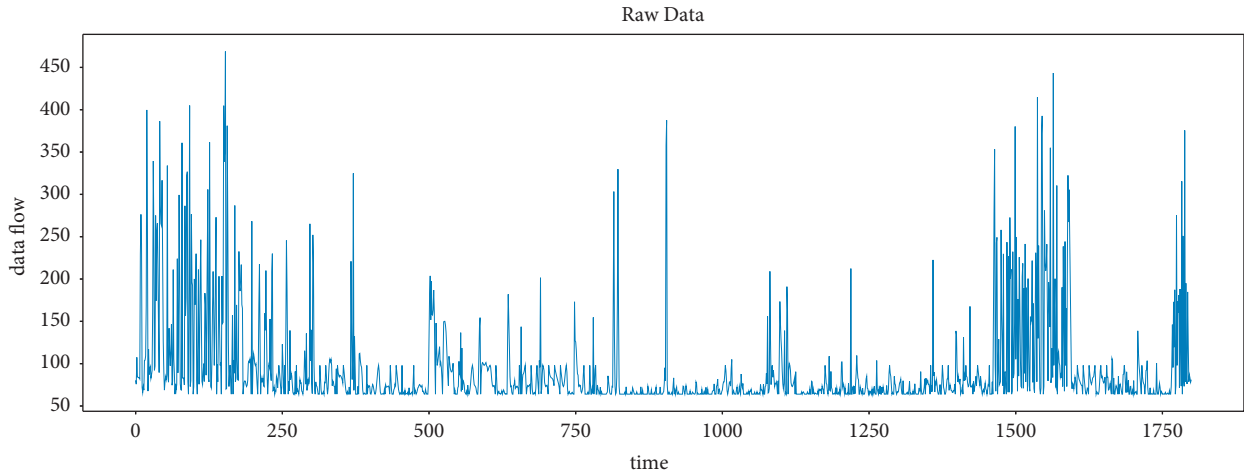


FIGURE 4: Network traffic data after null filling.

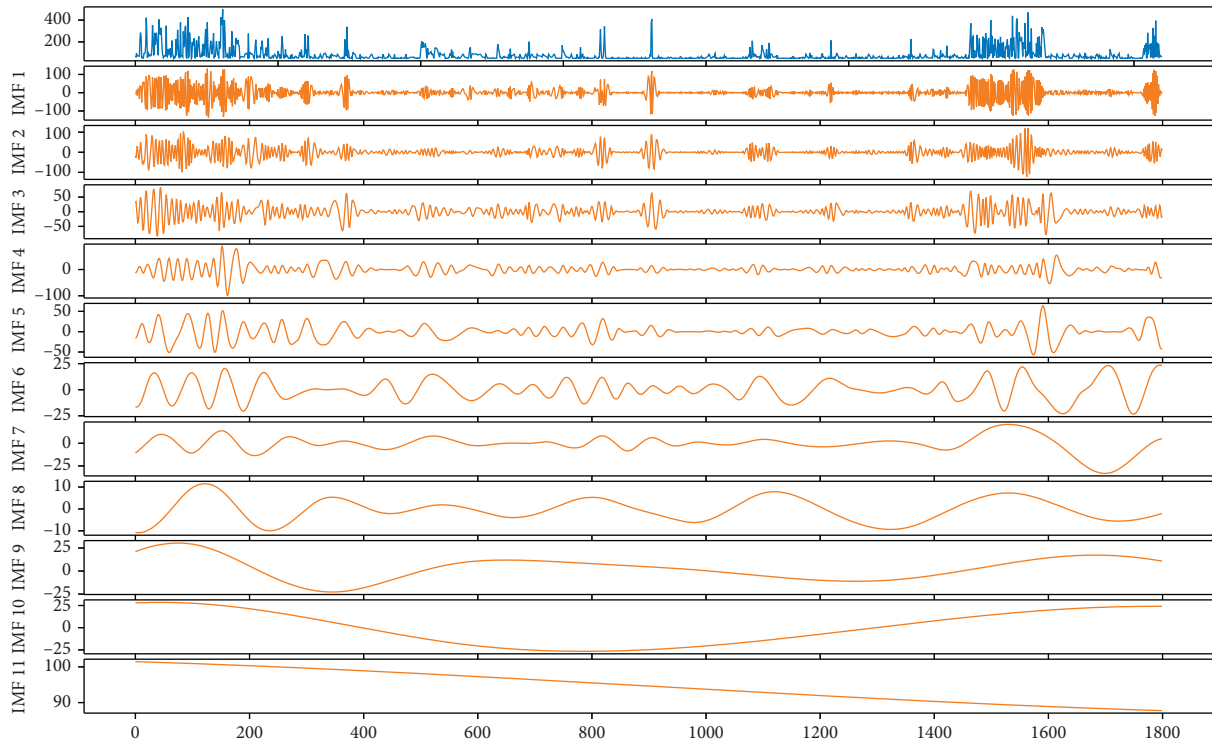


FIGURE 5: Network traffic data after decomposition.

2, respectively, the loss data of the model training process is shown in Figure 6.

(5) *Network traffic forecast.* 360 test data were predicted, and the first 100 predicted results were shown in Figure 7. TestOriginal_result represents the original data, and testPredict_result_101, testPredict_result_105, and testpredict_110, respectively, represent the prediction results when LN = 1, look_back takes 1, 5, and 10, respectively. TestPredict_result_201, testPredict_result_205, and testPredict_210, respectively, represent the prediction results when LN = 2, and look_back takes 1, 5, and 10, respectively.

(6) *Evaluate the prediction error of the model.* The LSTM model under different parameter combinations was executed for network traffic prediction, and the indexes of RMSE, MAE, and MAPE for each validation set were calculated. The results were shown in Table 1.

It can be seen from Table 1 that the prediction error of the model changes with the look_back increases, and the prediction error of single-layer LSTM and double-layer LSTM is different under the same look_back. Based on the above experiments, four groups of experiments were added, namely, when look_back takes 15 and 20, and multiple predictions were made in the case of single-layer network

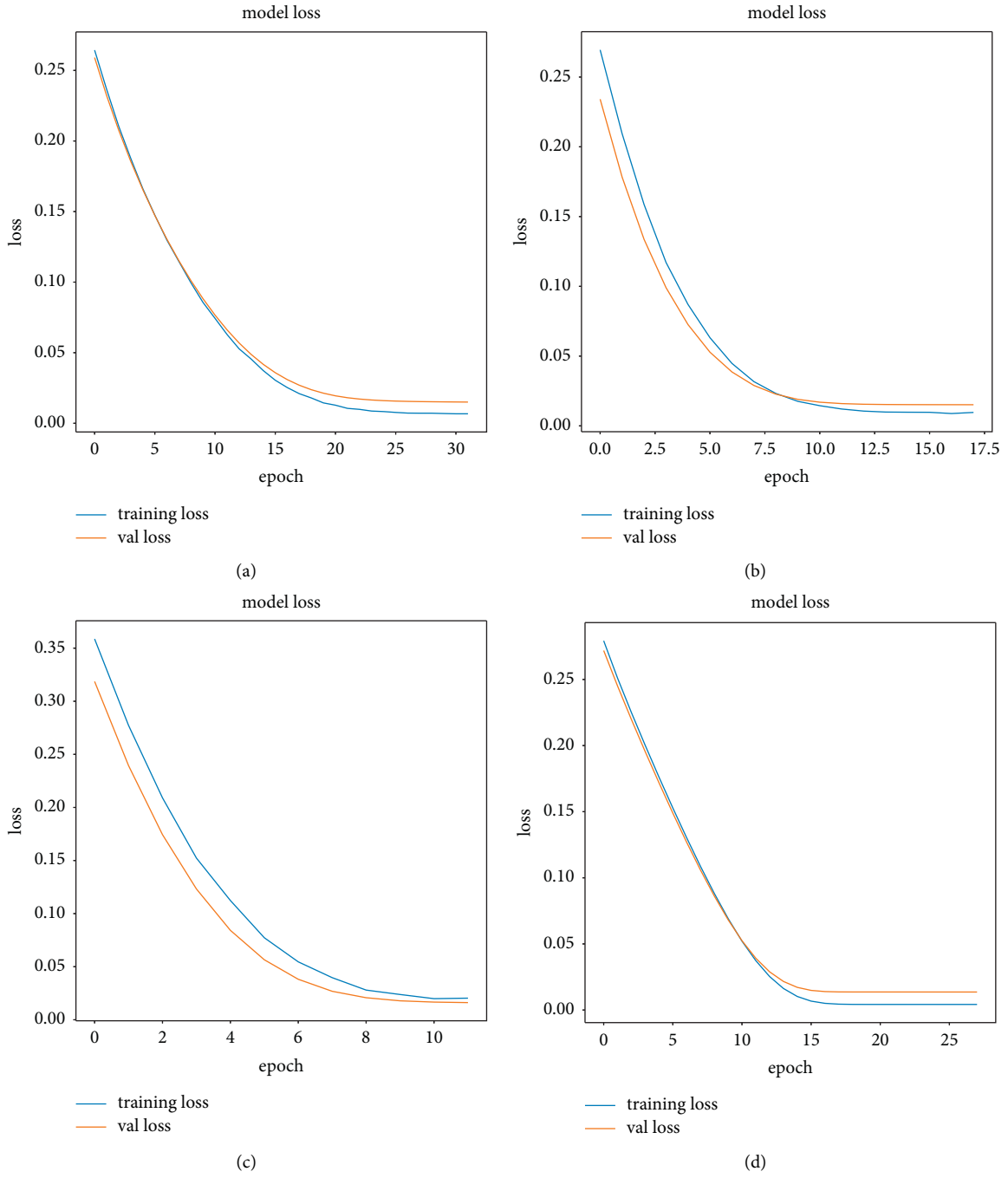


FIGURE 6: Continued.

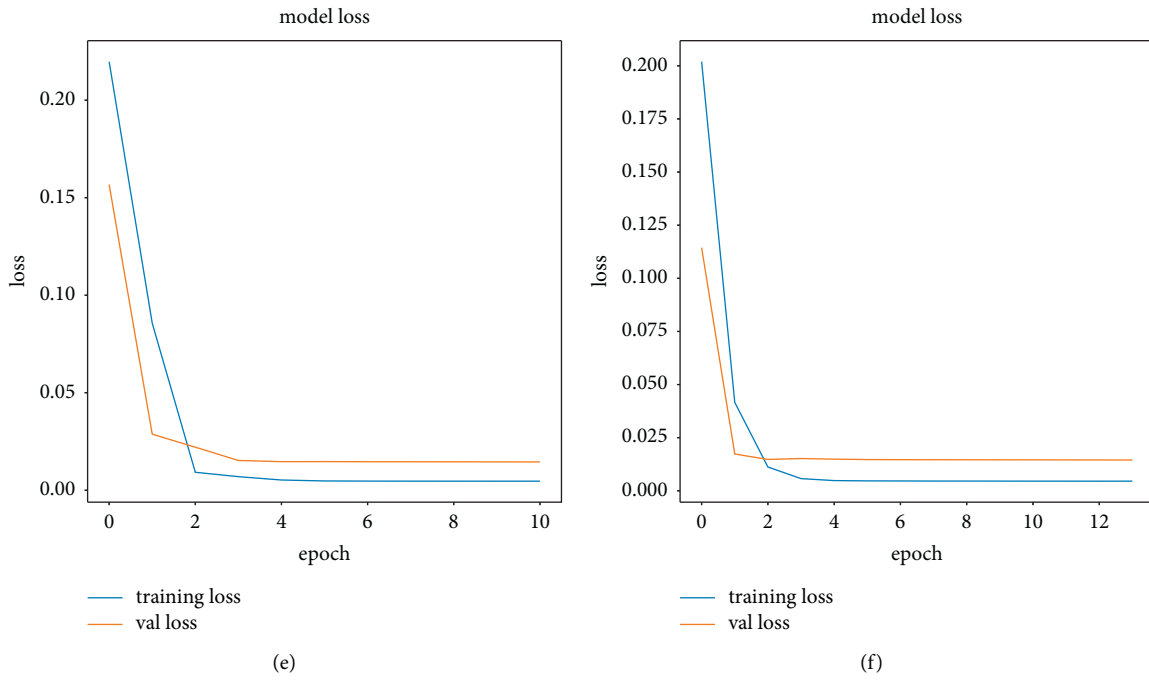


FIGURE 6: Model training loss under different parameter combinations.

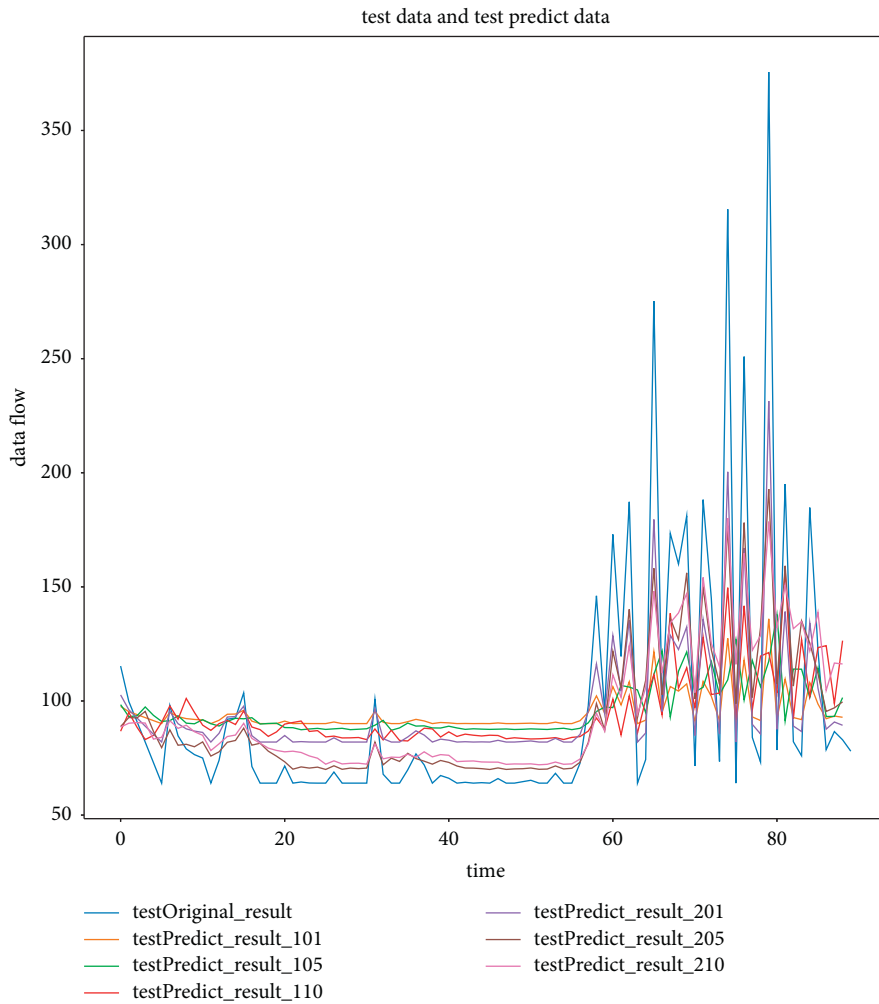


FIGURE 7: Test results of partial data.

TABLE 1: Errors of LSTM prediction models under different parameter combinations.

Model	Parameters	RMSE	MAE	MAPE (%)
LSTM101	(look_back, LN) = (1,1)	71.82	41.68	32.17
LSTM105	(look_back, LN) = (5, 1)	66.81	40.39	32.77
LSTM110	(look_back, LN) = (10,1)	67.08	41.58	34.11
LSTM201	(look_back, LN) = (1,2)	71.65	42.28	33.29
LSTM205	(look_back, LN) = (5,2)	69.58	40.57	31.19
LSTM210	(look_back, LN) = (10,2)	70.43	43.01	34.97

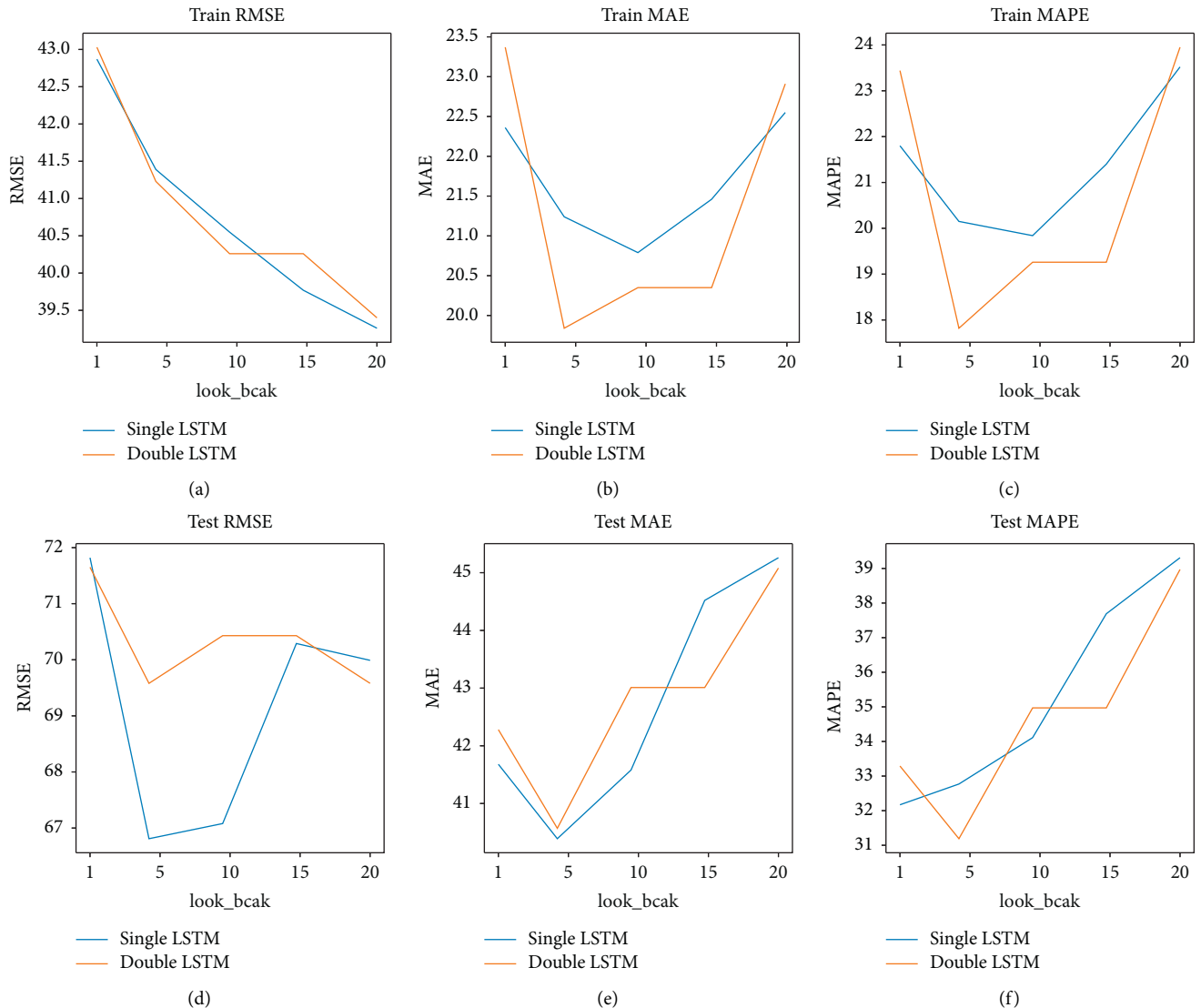


FIGURE 8: Errors of LSTM prediction model under different parameter combinations.

and double-layer network, respectively, and corresponding error values were calculated. The test results are shown in Figure 8.

It can be seen from Figure 8 that the setting of the number of hidden layers and the time step has a great impact on the fitting effect of LSTM. When a hidden layer is added, the prediction error changes, and the increase or decrease of

the prediction error is not fixed at different timesteps. When the time step is changed, that is, the look_back value is changed from small to large, and the trend of prediction error is also not fixed. For example, when the look_back value changes from 5 to 10, the prediction error of the single-layer LSTM model decreases, while the prediction error of the double-layer LSTM model increases.

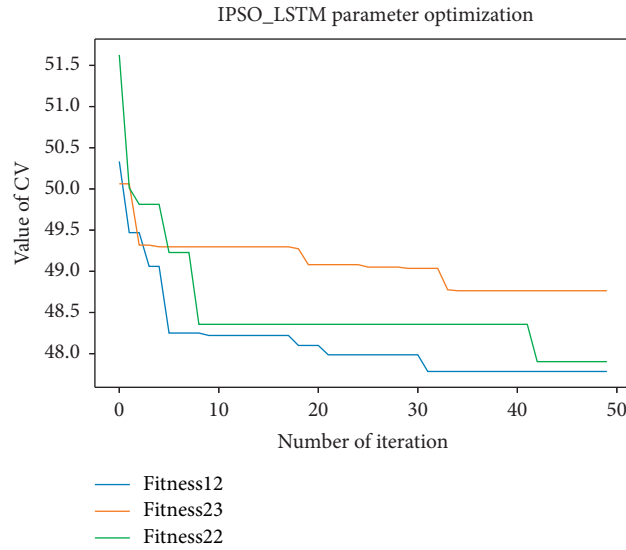


FIGURE 9: The change in fitness value.

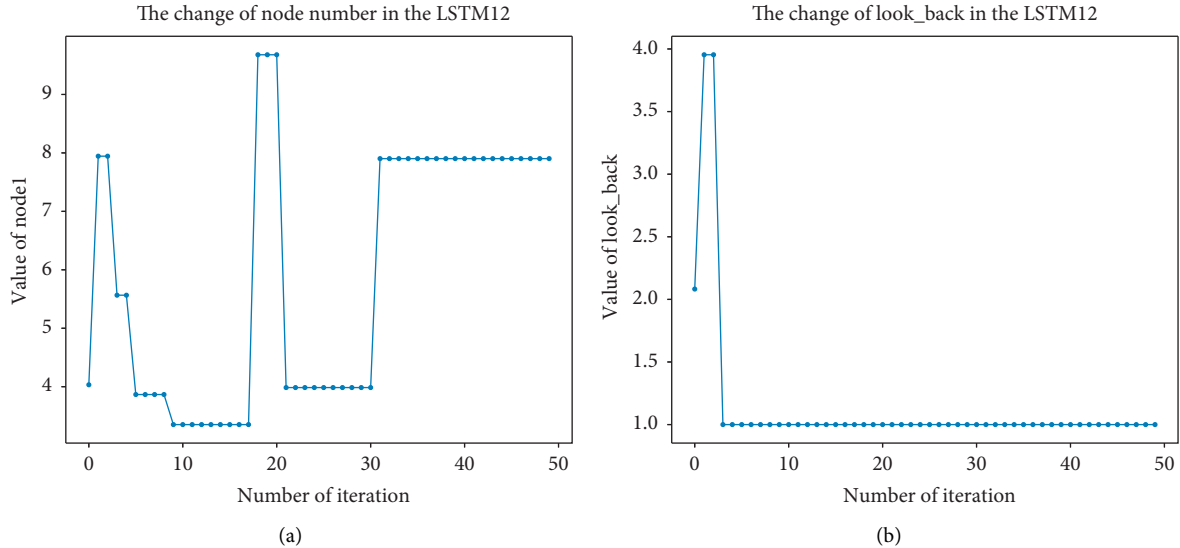


FIGURE 10: The change of parameter in LSTM12 during IPSO iteration (a) The change of node number. (b) The change of time step.

Therefore, for network traffic data, the prediction effect of the parameter combination set by the empirical method is unstable and cannot achieve the optimal prediction performance. Therefore, the Improved Particle Swarm Optimization (IPSO) will be adopted to carry out model optimization, that is, the intelligent algorithm will be used to efficiently obtain the parameter combination with the optimal prediction effect.

5.2.3. Parameter Optimization of LSTM Network Traffic Prediction Model Based on IPSO. The IPSO algorithm was used to optimize the LSTM network traffic prediction model, and parameters were optimized for single-layer LSTM and double-layer LSTM, respectively. The fitness value of the LSTM prediction model changed as the number of iterations increased during the experiment, as shown in Figure 9.

In Figure 9, fitness12, fitness23, and fitness22 correspond to the fitness values of the model IPSO-LSTM12 (2 parameters node1, lookback of the single layer), IPSO-LSTM23 (3 parameters node1, node2, lookback of the double layer), and IPSO-LSTM32 (2 parameters node1, lookback of double layer), respectively. The second parameter of IPSO-LSTM22 is set as node2 = 4 according to the optimization results of LSTM23.

It can be seen from Figure 9 that the final convergence value of fitness12 is less than fitness23 and fitness22, the convergence rate is faster than fitness23, and the fitness22 final convergence value is only slightly smaller than the fitness of 23. This shows that for the long-term prediction of network traffic data if the fitness value from a single hidden layer LSTM optimized by the particle swarm algorithm is slightly smaller than that from a two-layer hidden layer

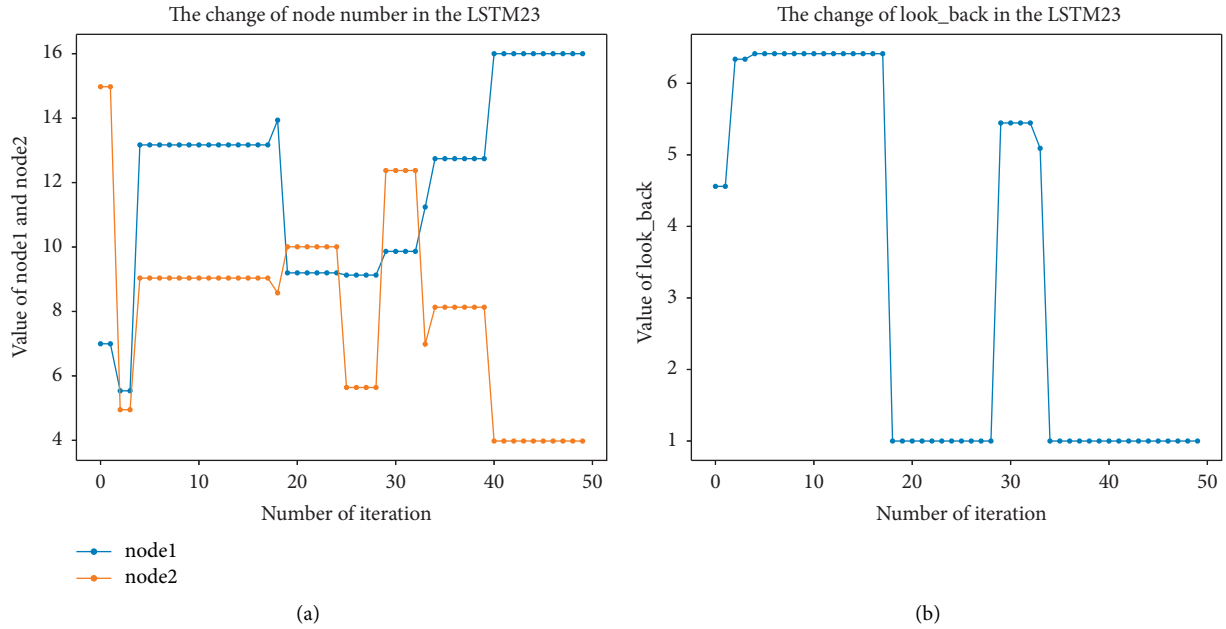


FIGURE 11: The change of parameter in LSTM23 during IPSO iteration. (a) The change of node number. (b) The change of time step.

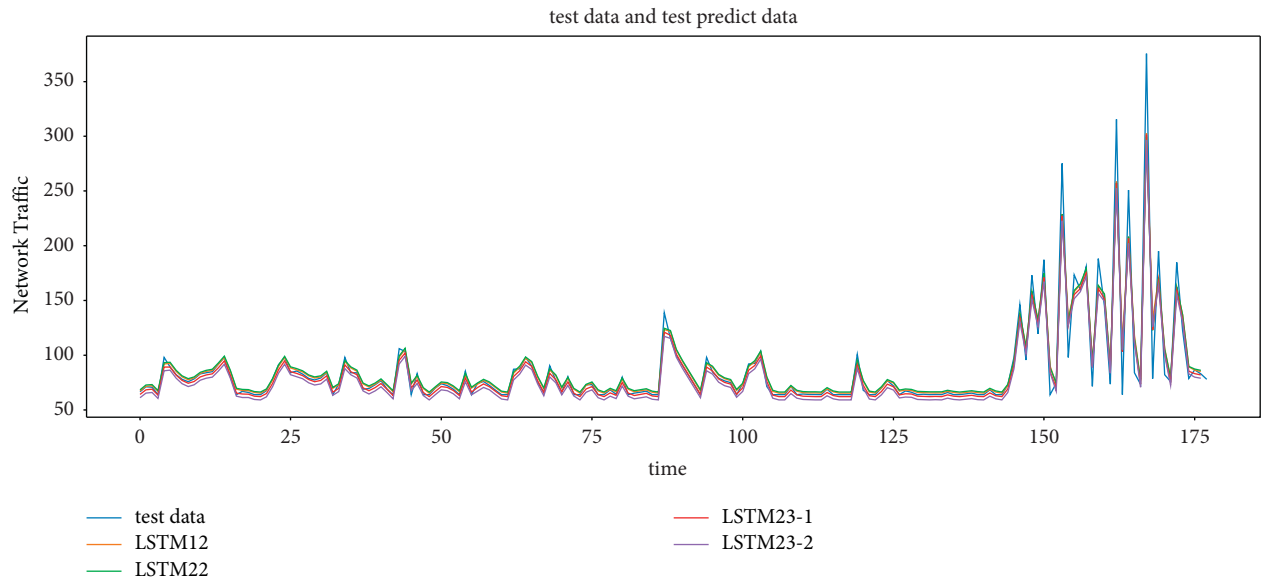


FIGURE 12: The comparison of prediction results of the four models.

LSTM optimized by the particle swarm algorithm, convergence speed is faster.

It can be seen that compared with the empirical method of setting LSTM parameters, the RMSE of the IPSO for setting LSTM parameters is reduced by 20%, which means that the IPSO algorithm can effectively find the optimal parameter combination of LSTM network traffic prediction and reduce the prediction error.

In addition, Figure 10 shows the changes in node number and time step size during the IPSO-LSTM12 model optimization that shows the process of the optimal parameter value of the LSTM network traffic model determined by the improved PSO algorithm.

It can be seen from Figure 10 that the optimal parameters of the LSTM12 model are set as node1 = 8 and look back = 1. Therefore, in the prediction of network traffic data used in this paper, the optimal configuration of the single-layer LSTM model is to set the number of neurons to 8 and the time step to 1.

The changes in node number and time step size in IPSO-LSTM23 model optimization are shown in Figure 11.

It can be seen from Figure 11 that the optimal parameters of LSTM23 model are set as node1 = 16, node2 = 4, and look back = 1. Therefore, in the prediction of network traffic data used in this paper, the optimal configuration of the two-layer LSTM model is to set the number of network neurons in the

TABLE 2: Errors of LSTM prediction models under different parameter combinations.

Index	IPSO-LSTM12	IPSO-LSTM22	IPSO-LSTM23-1	IPSO-LSTM23-2
RMSE	46.93	46.77	47.20	46.93
MAE	21.61	40.71	42.47	41.14
MAPE (%)	20.67	31.28	33.07	32.58

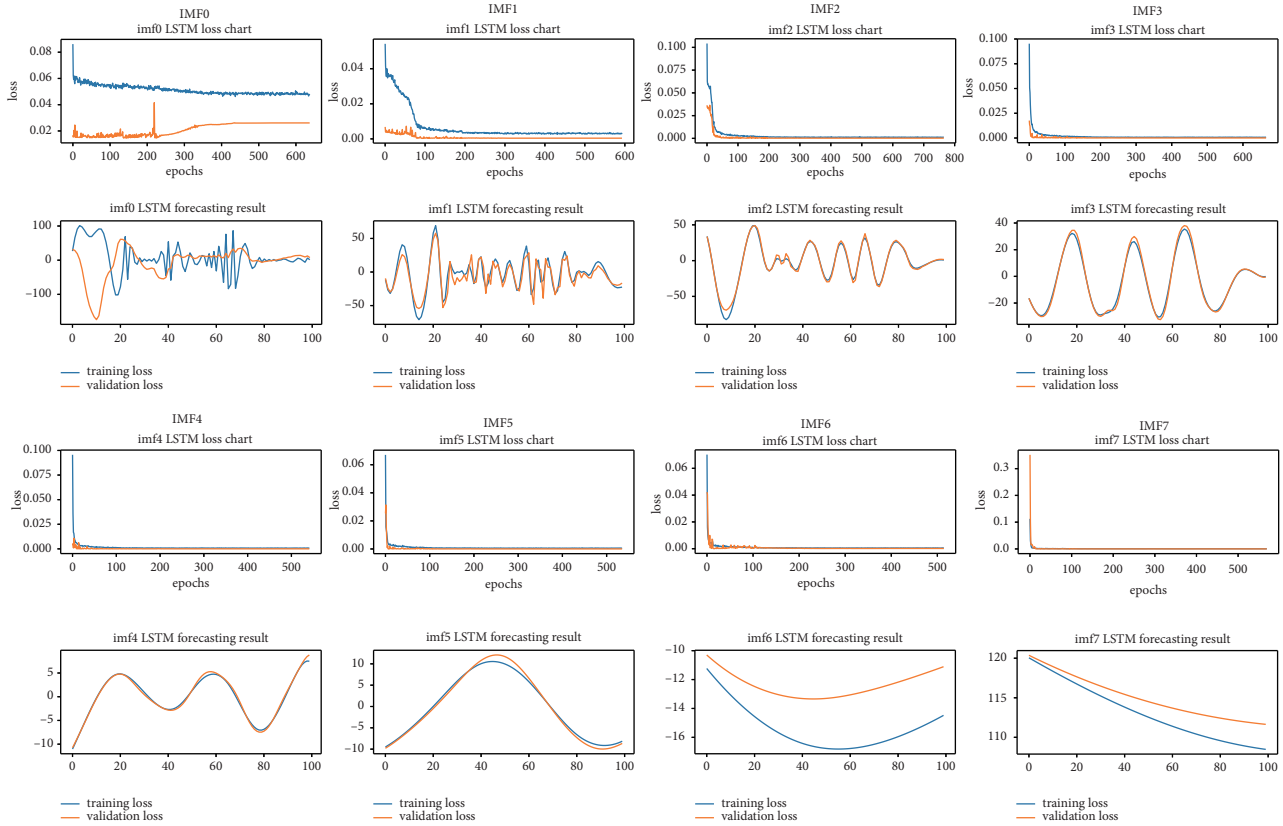


FIGURE 13: The predictive performance of each IMF.

first layer to 16, the number of neurons in the second layer to 4, and the time step length to 1.

To evaluate the prediction performance of the LSTM model after parameter optimization by IPSO, network traffic data samples at 180 time points are used for verification. In this paper, the IPSO optimized single-layer LSTM IPSO-LSTM12, double parameter optimization model IPSO-LSTM22 of double-layer LSTM, three parameters optimization model IPSO LSTM23-1 (no dropout in training) of double-layer LSTM, three-parameter optimization model of IPSO LSTM23-2 (dropout in training) of double-layer LSTM are compared, and Figure 12 shows the model prediction results for the last 180 test data.

It can be seen from Figure 12 that the prediction results of the LSTM model with different parameter combinations have a good fitting effect, and the prediction results of the single-layer LSTM dual-parameter optimization model IPSO-LSTM12 are better than those of other parameter configuration models. To compare the predictive performance of the four models more clearly, the predictive performance evaluation index values of the four models in

Figure 12 are obtained, respectively, and the results are shown in Table 2.

As it can be seen from Table 2, compared with single-layer LSTM12, two-layer LSTM22 has slightly fewer prediction errors in RMSE and MAE, while MAPE is slightly bigger. If only RMSE or MAE evaluation index is considered, LSTM22 is better than LSTM12, while only MAPE evaluation indicators are considered, LSTM12 is considered better than LSTM22. On the whole, the prediction error of LSTM12 and LSTM22 is less than that of the other three prediction models, that is, the prediction effect of LSTM12 and LSTM22 on network traffic data is better than that of the other three models. The prediction error of LSTM23-2 is less than that of LSTM23-1, which indicates that the optimization of dropout parameters added in the training reduces the prediction error of the model and improves the prediction performance of the model.

5.2.4. Network Traffic Prediction Based on CEEMDAN-IPSO-LSTM. Through testing on a 500-time data set, the predictive performance of each IMF is shown in Figure 13.

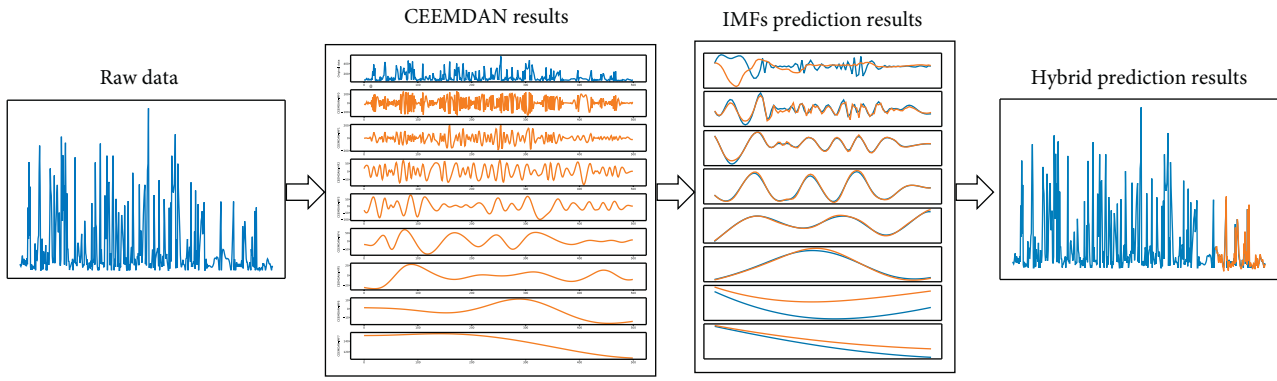
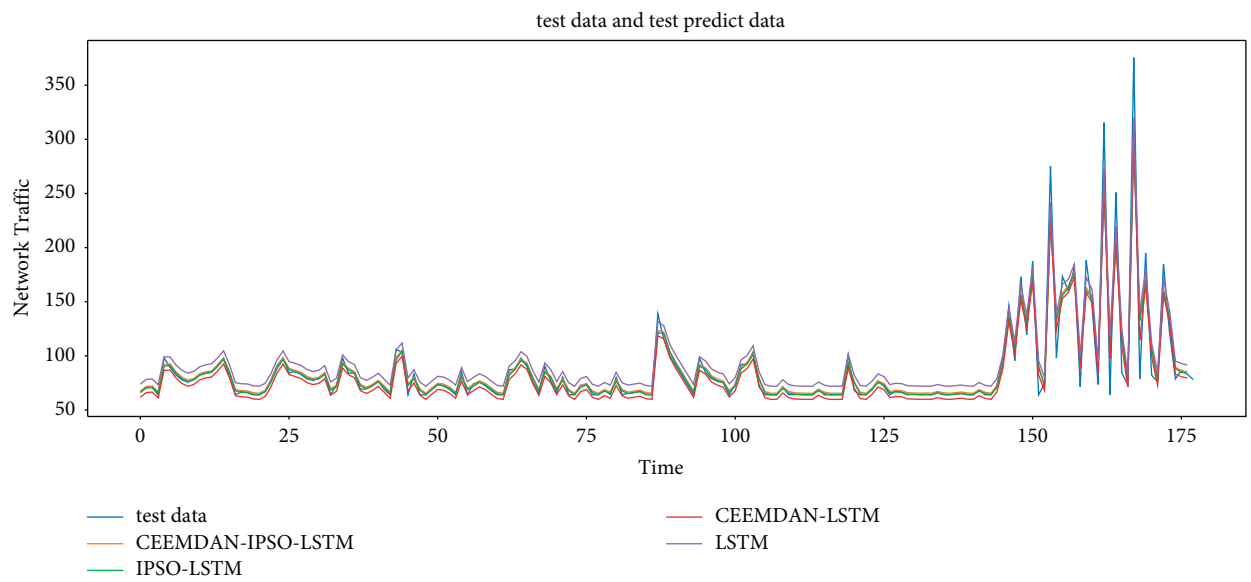
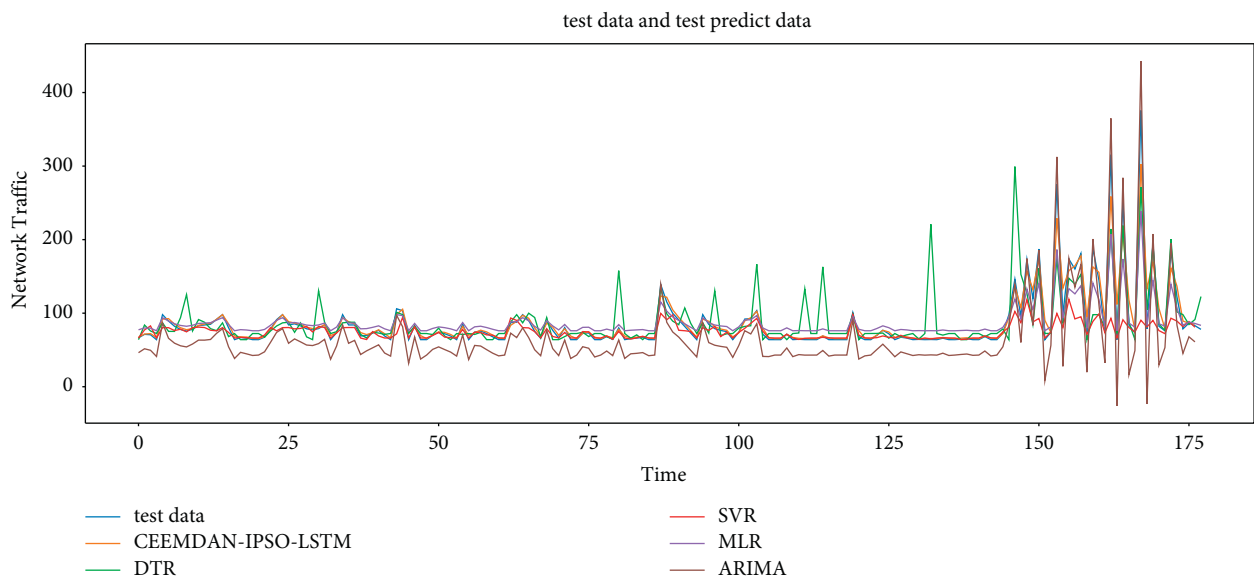


FIGURE 14: The CEEMDAN-IPSO-LSTM forecasting flowchart.



(a)



(b)

FIGURE 15: The comparison of prediction results of the eight models (a) Prediction results of CEEMDAN-IPSO-LSTM and other neural network methods (b) Prediction results of CEEMDAN-IPSO-LSTM and other regression methods.

TABLE 3: The prediction error with five classical models.

Index	CEEMDAN-IPSO-LSTM	CEEMDAN-LSTM	IPSO-LSTM	LSTM	ARIMA	SVR	DTR	MLR
RMSE	46.93	51.77	47.20	52.53	90.22	65.59	89.78	71.74
MAE	21.61	27.61	42.47	41.14	53.98	37.58	51.42	41.52
MAPE (%)	20.67	27.94	33.07	32.58	43.79	25.61	49.15	32.19

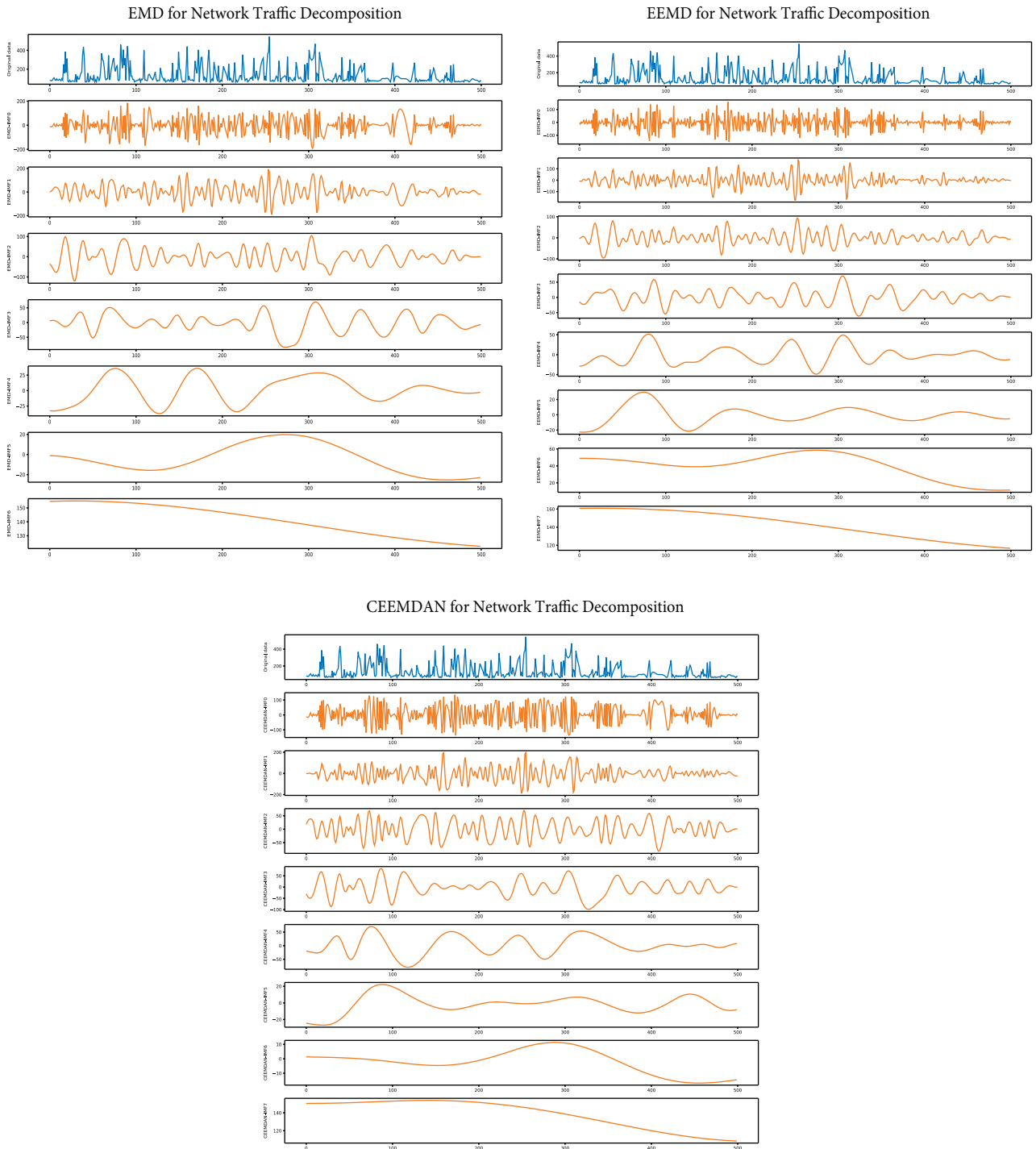


FIGURE 16: The decomposition results of three decomposition methods.

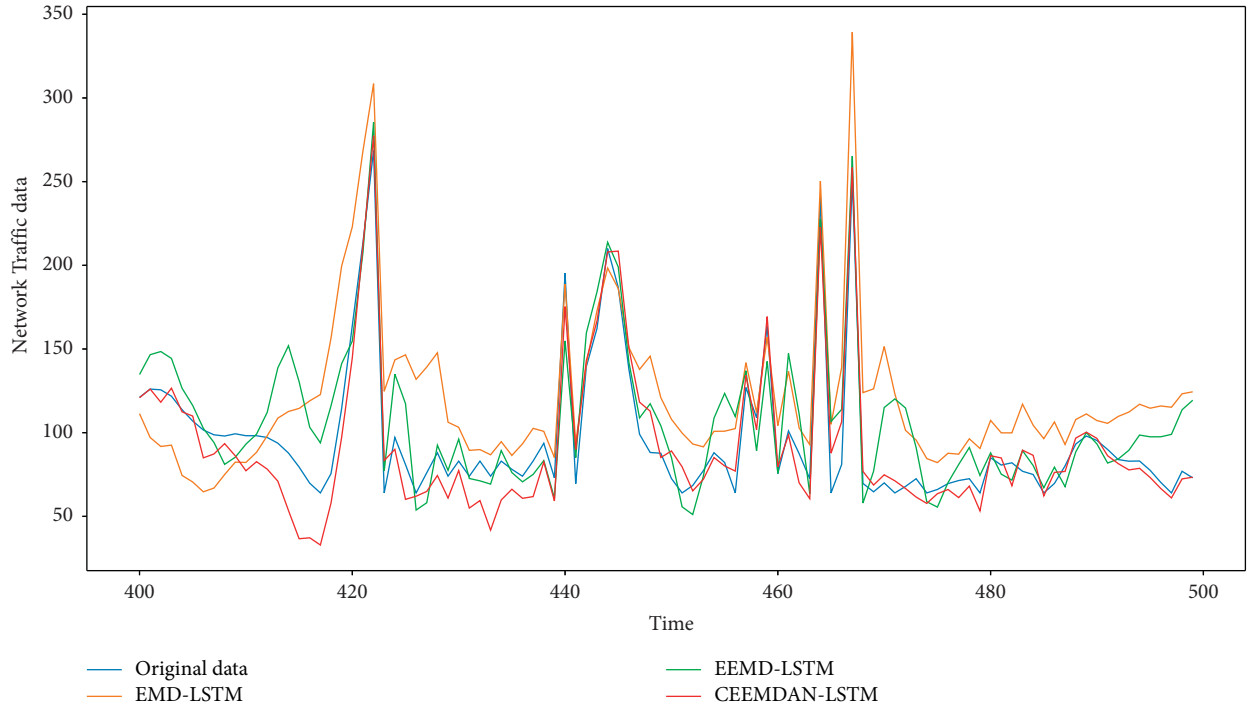


FIGURE 17: The predictions by LSTM combining the three decomposition methods.

Figure 13 shows the prediction results and training loss of eight IMFs and it has a better prediction effect. IMF0 and IMF7 are a little poor, in which the loss of the training set is very high during the whole training process. Especially, the loss of IMF0 is relatively large. For the remaining IMFs, LSTM predicts them well. Despite this problem, the overall results were excellent when the predictions were integrated.

After finishing predicting all IMFs, the final prediction result is integrated by superimposing the predicted results of each IMF. Figure 14 shows the forecasting flowchart of CEEMDAN-IPSO-LSTM.

5.3. Result Analysis. To evaluate the prediction effect of the proposed hybrid method CEEMDAN-IPSO-LSTM, it is compared with other neural network prediction methods like CEEMDAN-LSTM, IPSO-LSTM, and LSTM, and other predictive models like ARIMA, Support Vector Regression (SVR), Decision Tree Regressor (DTR), and Multivariate Linear Regression (MLR). Similarly, the network traffic data samples at 180-time points were used for verification, and the prediction results of the eight models are shown in Figure 15.

Figure 15 shows that the prediction effects of different models and the hybrid prediction model have a better fitting effect which indicates that the prediction results of the CEEMDAN-IPSO-LSTM model are better than those of other models. To compare the prediction performance of the eight models more clearly, their predictive performance evaluation index values were obtained, respectively, and shown in Table 3.

It can be seen from Table 3, that the prediction errors of the LSTM-based model are all less than regression

prediction models, which indicates that the LSTM network traffic prediction model has a better prediction effect than other regression network traffic prediction models. In other words, the LSTM is more suitable for solving long-term network traffic data prediction and processing real-time variability of network traffic data. In addition, the RMSE, MAE, and MAPE index values of the CEEMDAN-IPSO-LSTM prediction model are all smaller than other neural network prediction models, indicating that the proposed hybrid model CEEMDAN-IPSO-LSTM is better than other prediction models in network traffic prediction.

Besides, we make comparisons of decomposition methods like EMD, EEMD, and CEEMDAN. Firstly, based on a 500-time network flow data, we decompose the original data into several IMFs and compare the decomposition results of three decomposition methods. Then, we make predictions by LSTM methods combining the three decomposition methods to explain which method works better.

In Figure 16, there are seven IMFs of EMD, eight of EEMD, and eight of CEEMDAN, including residue. We only know that different decomposition results make the prediction accuracy different, but it is hard to see which one produces the better prediction. So, we make predictions by LSTM combining the three decomposition methods and the results are in Figure 17.

Figure 17 shows the red lines fit the raw data more closely, which shows that the predicted result of CEEMDAN-LSTM is closer to the real value. To further verify the effect of different decomposition methods, Table 4 gives the prediction error of CEEMDAN-LSTM, EEMD-LSTM, and EMD-LSTM.

TABLE 4: The prediction error of LSTM combining different decomposition methods.

Index	CEEMDAN-LSTM	EEMD-LSTM	EMD-LSTM
RMSE	14.12	22.74	36.66
MAE	10.94	17.46	30.65
MAPE (%)	15.69	16.35	26.05

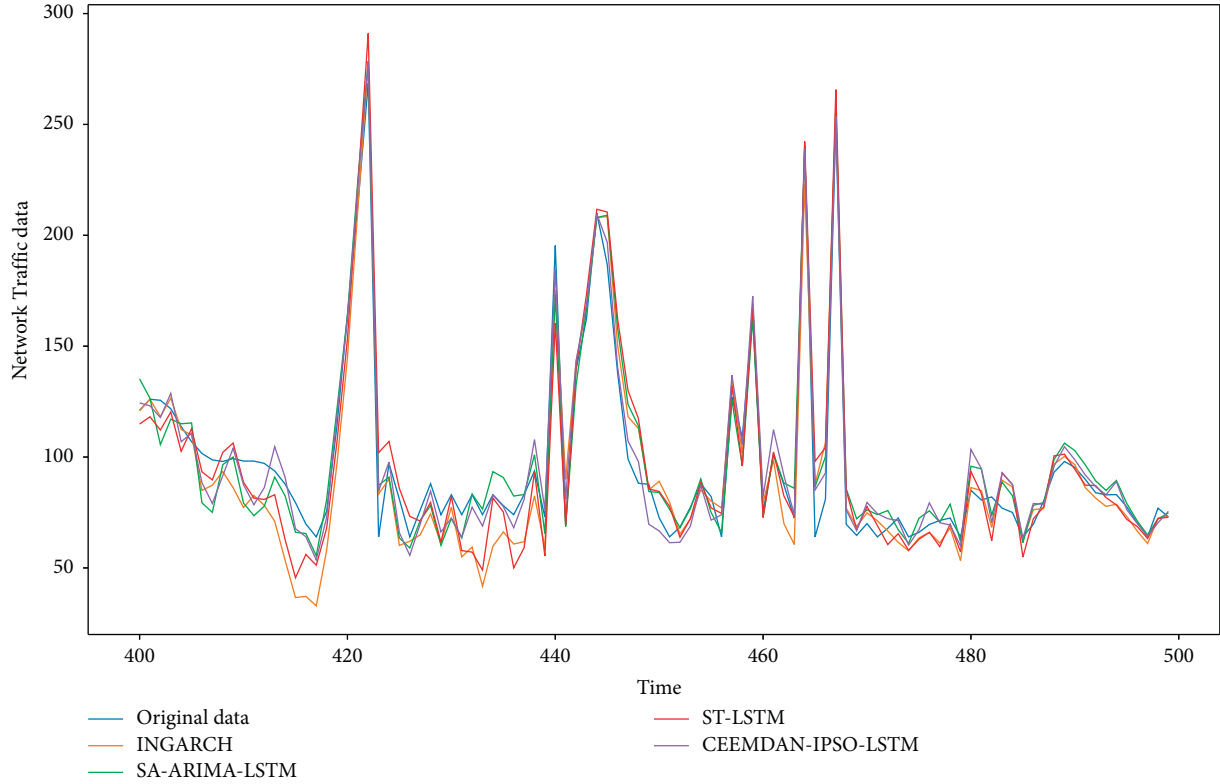


FIGURE 18: The prediction results of the state-of-the-art models.

TABLE 5: The prediction error of the state-of-the-art models.

Index	CEEMDAN-IPSO-LSTM	SA-ARIMA-LSTM	ST-LSTM	INGARCH
RMSE	8.59	10.79	13.45	14.12
MAE	6.9	8.23	10.00	10.94
MAPE (%)	8.02	9.18	12.20	15.69

In Table 4, the prediction error of CEEMDAN-LSTM is significantly less than the other two methods, which indicates CEEMDAN can decompose data more effectively so that LSTM can predict better. That is to say, the results verify the superiority of CEEMDAN for data decomposition.

Also, based on the same 500-time network flow data, we compare CEEMDAN-IPSO-LSTM with three state-of-the-art prediction models to verify the effectiveness of the proposed network traffic prediction model, like ST-LSTM, SA-ARIMA-BPNN, and INGARCH. The last 100-time prediction data of the four methods are in Figure 18.

In Figure 18, the four prediction methods do a good job of forecasting network traffic. Figure 18 shows that the purple

and green lines match the raw data represented by the blue lines better, which demonstrates that the proposed method and the SA-ARIMA-LSTM make more effective predictions close to reality. To compare the prediction accuracy of the four methods more clearly, the prediction error of the four methods is calculated similarly and shown in Table 5.

In the same appearance as Figure 18, CEEMDAN-LSTM has the lowest prediction error. The appearances of Figure 18 and Table 5 prove the superiority of the CEEMDAN-IPSO-LSTM in this paper once again.

Above all, the CEEMDAN-IPSO-LSTM has a better prediction effect and higher reliability for the future prediction of network traffic.

6. Conclusion and Future Work

Network traffic prediction can be applied to network resource optimization and network congestion avoidance, which makes great significance for network business planning, data management, fault detection, resource allocation, and other operations. In this paper, a hybrid deep interval prediction model has been proposed for network traffic forecasting to improve the prediction accuracy. Firstly, the nonparametric LSTM neural network is used to establish the network traffic prediction model, and the Improved Particle Swarm Optimization algorithm is used to optimize the hyperparameters of the established LSTM prediction model, and further obtain the optimized LSTM network prediction model—IPSO-LSTM12, IPSO-LSTM23 and IPSO-LSTM32—which reduces the RMSE by 20% compared to the Experience-based LSTM. Besides, the prediction performance of single-layer LSTM is better than double-layer LSTM in network traffic prediction. Then the CEEMDAN is introduced to decompose the network traffic time series into different modes to reduce the complexity of the network traffic sequence. To verify the effectiveness of the proposed models, the proposed CEEMDAN-IPSO-LSTM model is applied to network traffic prediction and compared with other neural network prediction methods and regression methods. The experimental results show that compared with other prediction models and the traditional LSTM model, the CEEMDAN-IPSO-LSTM model reduces the prediction error and obtains a better fitting effect, which demonstrates that the proposed hybrid method improves network traffic prediction accuracy.

In future work, we plan to enhance the prediction model from two aspects to further improve the prediction accuracy of network traffic. On the one hand, in the data pre-processing stage, we will try other data decomposition methods, such as Variational Mode Decomposition (VMD), wavelet packet, and combination method, to improve the stability and regularity of network traffic data decomposition. On the other hand, we will focus more on the error correction strategy of the hybrid model of network traffic forecasting, such as analysis of different error correction strategies, or re-decompose the IMF data, to enhance the prediction performance.

Data Availability

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

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This work was funded by the National Natural Science Foundation of China under Grant No. 62072363, the National Natural Science Foundation of China under Grant No. 61672416, Industrialization projects of the Education

Department of Shaanxi Province under Grant No. 21JC017, Industrial research project of Science and Technology Department of Shaanxi Province under Grant No. 2020GY-012, and Science and Technology Plan Project of Yulin under Grant No. 2019-175.

References

- [1] S. Iqbal, K. N. Qureshi, F. Shoaib, A. Ahmad, and G. Jeon, "Minimize the delays in software defined network switch controller communication," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 13, p. e5940, 2020.
- [2] L. Liu, S. Guo, G. Liu, and Y. Zeng, "Priority based online flow scheduling for network throughput maximization in software defined networking," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 9, p. e5633, 2020.
- [3] M. Gupta and A. Sinha, "Distributed Temporal Data Prediction Model for Wireless Sensor Network," *Wireless Personal Communications*, vol. 119, no. 4, pp. 3699–3717, 2021.
- [4] W. Jiang, "Cellular traffic prediction with machine learning: A survey," *Expert Systems with Applications*, vol. 201, Article ID 117163, 2022.
- [5] C. Pan, Y. Wang, H. Shi, J. Shi, and R. Cai, "Network Traffic Prediction Incorporating Prior Knowledge for an Intelligent Network," *Sensors*, vol. 22, no. 7, p. 2674, 2022.
- [6] N. Jiang, Y. Deng, O. Simeone, and A. Nallanathan, "Online Supervised Learning for Traffic Load Prediction in Framed-ALOHA Networks," *IEEE Communications Letters*, vol. 23, no. 10, pp. 1778–1782, 2019.
- [7] Y. Ji, Y. Wu, D. Zhang et al., "A Novel Flash P2P Network Traffic Prediction Algorithm based on ELMD and Garch," *International Journal of Information Technology and Decision Making*, vol. 19, no. 01, pp. 127–141, 2020.
- [8] C. Katris and S. Daskalaki, "Dynamic bandwidth allocation for video traffic using FARIMA-based forecasting models," *Journal of Network and Systems Management*, vol. 27, no. 1, pp. 39–65, 2019.
- [9] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu, "Predicting Inter-Data-Center Network Traffic Using Elephant Flow and Sublink Information," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 1–792, 2016.
- [10] L. Nie, X. Wang, S. Wang et al., "Network Traffic Prediction in Industrial Internet of Things Backbone Networks: A Multitask Learning Mechanism," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7123–7132, 2021.
- [11] M. Kim, "Network traffic prediction based on INGARCH model," *Wireless Networks*, vol. 26, no. 8, pp. 6189–6202, 2020.
- [12] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in *Proceedings of the 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pp. 324–328, IEEE, Wuhan, China, November 2016.
- [13] L. Mingyan and T. Sucheng, "Large Scale Network Data Flow Parallel Forecasting Method Simulation," *Computer Simulation*, vol. 35, no. 08, pp. 206–209, 2018.
- [14] H. Yang, X. Li, W. Qiang, Y. Zhao, W. Zhang, and C. Tang, "A network traffic forecasting method based on SA optimized ARIMA-BP neural network," *Computer Networks*, vol. 193, Article ID 108102, 2021.
- [15] S. Hu and Z. Yuxue, "Network Traffic Modeling and Forecasting based on ARIMA," *Communications Technology*, vol. 52, no. 04, pp. 903–907, 2019.

- [16] C. Katris and S. Daskalaki, "Comparing forecasting approaches for Internet traffic," *Comparing forecasting approaches for Internet traffic Expert Systems with Applications*, vol. 42, no. 21, pp. 8172–8183, 2015.
- [17] C. Xiang, Q. Peixin, and Q. Xilong, "Network Traffic Prediction Based on MK-SVR," *Journal of Information and Computational Science*, vol. 12, no. 8, pp. 3185–3197, 2015.
- [18] X. Chen, Y. Liu, and J. Zhang, "Traffic prediction for Internet of Things through support vector regression model," *Internet Technology Letters*, vol. 5, no. 3, p. e336, 2022.
- [19] X. Zheng, W. Lai, H. Chen, and S. Fang, "Data Prediction of Mobile Network Traffic in Public Scenes by SOS-vSVR Method," *Sensors*, vol. 20, no. 3, pp. 603–619, 2020.
- [20] T. De Schepper, M. Camelo, J. Famaey, and S. Latre, "Traffic classification at the radio spectrum level using deep learning models trained with synthetic data," *International Journal of Network Management*, vol. 17, no. 4, p. 30, 2020.
- [21] W. Yong, Z. Huiyi, F. Hao, Y. Miao, and E. Wenlong, "Network traffic classification method basing on CNN," *Journal on Communications*, vol. 39, no. 01, pp. 14–23, 2018.
- [22] S. Nihale, S. Sharma, L. Parashar, and S. Upendra, "Network traffic prediction using long short-term memory," in *Proceedings of the 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 338–343, IEEE, Coimbatore, India, July 2020.
- [23] W. Shihao, Z. Qinzhen, Y. Han, and L. Qiamnu, "A Network Traffic Prediction Method Based on LSTM," *ZTE Communications*, vol. 17, no. 2, pp. 19–25, 2019.
- [24] L. Xiaolin and W. Teng, "Efficient network traffic prediction method based on PF-LSTM network," *Application Research of Computers*, vol. 36, no. 2, pp. 3833–3836, 2019.
- [25] Z. Tian and P. Song, "A novel network traffic combination prediction model," *International Journal of Communication Systems*, vol. 35, no. 7, p. e5097, 2022.
- [26] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2353–2358, IEEE, Udupi, India, September 2017.
- [27] G. Fang, C. Lei, and Y. Ziwen, "Real-time traffic prediction based on MGU for large-scale IP backbone networks," *Journal of Shandong University*, vol. 49, no. 2, pp. 88–95, 2019.
- [28] X. Pang, Y. Zhou, P. Wang, W. Lin, and V. Chang, "An innovative neural network approach for stock market prediction," *The Journal of Supercomputing*, vol. 76, no. 3, pp. 2098–2118, 2020.
- [29] Y. L. Zhou, R. J. Han, Q. Xu, Q. Jiang, and W. Zhang, "Long short term memory networks for CSI300 volatility prediction with Baidu search volume," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 10, Article ID e4721, 2019.
- [30] B. Shao, M. Li, Y. Zhao, and G. Bian, "Nickel Price Forecast Based on the LSTM Neural Network Optimized by the Improved PSO Algorithm," *Mathematical Problems in Engineering*, vol. 2019, Article ID 1934796, 15 pages, 2019.
- [31] Z. Jianhui, D. Ting, and C. Bo, "AQI Prediction Based on Long Short-Term Memory Model with Spatial-Temporal Optimizations and Fireworks Algorithm," *Journal of Wuhan University (Natural Science Edition)*, vol. 65, no. 3, pp. 250–262, 2019.
- [32] Z. Dianfan, L. Zihao, and C. Shuhong, "Research on Model Temperature Prediction Method Based on Fireworks Algorithm and Long and Short Time Memory Network," *Acta Metrologica Sinica*, vol. 41, no. 006, pp. 640–645, 2020.
- [33] A. Guo, A. Jiang, J. Lin, and X. Li, "Data mining algorithms for bridge health monitoring: Kohonen clustering and LSTM prediction approaches," *The Journal of Supercomputing*, vol. 76, no. 2, pp. 932–947, 2020.
- [34] L. Lian and Z. Tian, "Network traffic prediction model based on ensemble empirical mode decomposition and multiple models," *International Journal of Communication Systems*, vol. 34, no. 17, Article ID e4966, 2021.
- [35] J. Bi, X. Zhang, H. Yuan, J. Zhang, and M. Zhou, "A Hybrid Prediction Method for Realistic Network Traffic With Temporal Convolutional Network and LSTM," *IEEE Transactions on Automation Science and Engineering*, vol. 19, pp. 1869–1879, 2021.
- [36] G. Zhang, L. Xu, and Y. Xue, "The time dependency predictive model on the basis of community detection and long short term memory," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, Article ID e4184, 2017.
- [37] X. Ran, X. Zhou, M. Lei, W. Tepsan, and W. Deng, "A Novel K-Means Clustering Algorithm with a Noise Algorithm for Capturing Urban Hotspots," *Applied Sciences*, vol. 11, no. 23, Article ID 11202, 2021.
- [38] Z. H. Zhang, F. Min, G. S. Chen, S. Peng, and Z. Cheng, "Tripartition state alphabet-based sequential pattern for multivariate time series," *Cognitive Computation*, vol. 18, pp. 1–19, 2021.
- [39] W. Deng, X. Zhang, Y. Zhou et al., "An enhanced fast non-dominated solution sorting genetic algorithm for multi-objective problems," *Information Sciences*, vol. 585, pp. 441–453, 2022.
- [40] H. Cui, Y. Guan, H. Chen, and W. Deng, "A Novel Advancing Signal Processing Method Based on Coupled Multi-Stable Stochastic Resonance for fault detection," *Applied Sciences*, vol. 11, no. 12, p. 5385, 2021.
- [41] L. Wan, F. Fenling, and J. Qiwei, "Prediction for railway passenger volume based on modified PSO optimized LSTM neural network," *Journal of Railway Science and Engineering*, vol. 15, no. 12, pp. 3274–3280, 2018.
- [42] H. Cui, Y. Guan, and H. Chen, "Rolling Element Fault Diagnosis Based on VMD and Sensitivity MCKD," *IEEE Access*, vol. 9, Article ID 120308, 2021.
- [43] F. Zhou, Z. Huang, and C. Zhang, "Carbon price forecasting based on CEEMDAN and LSTM," *Applied Energy*, vol. 311, Article ID 118601, 2022.

Research Article

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

Aiping Zhou ^{1,2} and Jin Qian ¹

¹School of Information Engineering, Taizhou University, Taizhou, China

²Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, China

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

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

Academic Editor: Weiwei Liu

Copyright © 2022 Aiping Zhou and Jin Qian. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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

1. Introduction

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

The cardinality of a node indicates the number of distinct other nodes it communicates with. We consider two types of node cardinalities: source cardinality and destination cardinality, where source cardinality indicates the number of distinct destinations that a node connects to and destination cardinality indicates the number of distinct sources that a

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

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

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

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

There have been many efforts on cardinality estimation. A simple method is to maintain distinct connections between any two hosts in the network [25, 26]. The method can measure the cardinality of each host accurately, but it is not practical to process massive network traffic due to large memory overhead [27, 28]. Sampling-based methods [29, 30] and data stream-based methods [31–33] are proposed to tackle the challenges caused by big network traffic. However, the estimation accuracy of sampling-based methods depends on the sampling rate, that is, they maintain a small number of distinct connections at the small sampling rate, but the accuracy of cardinality estimation decreases; on

the contrary, the accuracy of cardinality estimation increases at the high sampling rate, but the memory overhead also increases. Data stream-based methods demonstrate great superiority on memory utilization and measurement accuracy, where various types of summary data structures have been widely used in traffic measurement due to the excellent compression efficiency and acceptable accuracy.

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

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

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

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

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

2. Related Work

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

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

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

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

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

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

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

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

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

3. Problem Formulation

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

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

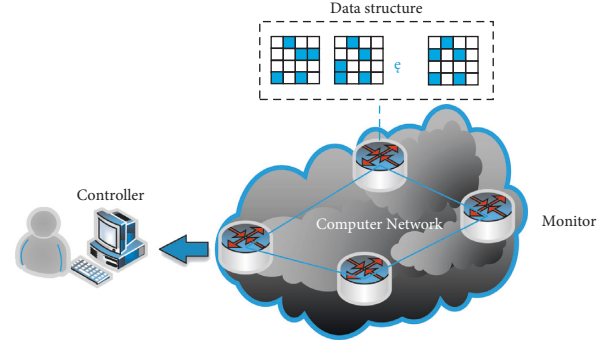


FIGURE 1: The distributed monitoring system.

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

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

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

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

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

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

4. Our Algorithm

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

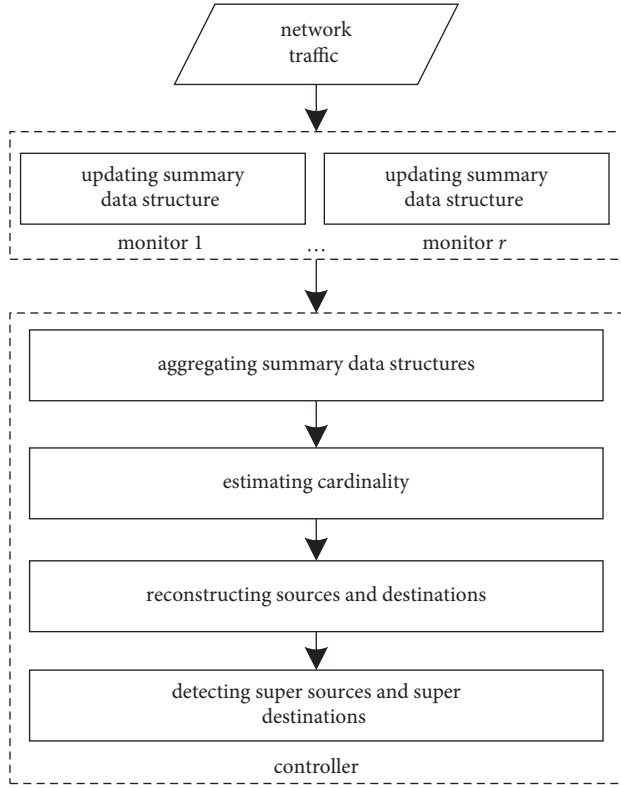


FIGURE 2: Framework of our method.

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

$$B = (B_1, B_2, \dots, B_H). \quad (3)$$

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

$$f_i: \{0, 1, \dots, N-1\} \longrightarrow \{0, 1, \dots, n_i-1\}, \quad (4)$$

$$h_i: \{0, 1, \dots, M-1\} \longrightarrow \{0, 1, \dots, m_i-1\}, \quad (5)$$

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

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

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

$$h_i(x) \equiv c'_i \pmod{m_i}, \quad 1 \leq i \leq H, \quad (7)$$

where c_i and c'_i are the values of modulus operation, n_1, n_2, \dots, n_H are selected as pair-wise coprime integers to make the

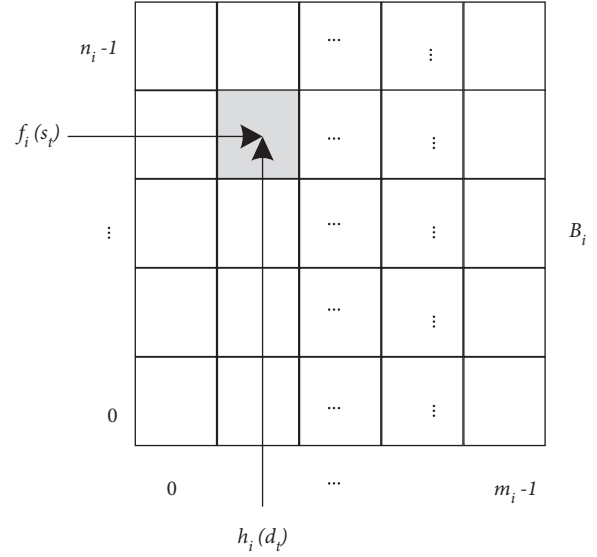


FIGURE 3: Data structure.

summary data structure reversible, and m_1, m_2, \dots, m_H are also pair-wise coprime integers.

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

$$B^r = (B_1^r, B_2^r, \dots, B_H^r), \quad r = 1, 2, \dots, R. \quad (8)$$

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

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

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

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

$$B_i[j][k] = B_i^1[j][k] \oplus \dots \oplus B_i^R[j][k], \quad (10)$$

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

ALGORITHM 1: Updating summary data structures.

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

ALGORITHM 2: Estimating cardinality.

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

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

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

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

If other sources are not mapped to the $f_i(s)$ th row, the estimated source cardinality is very close to its real value. Actually, each $B_i(s)$ ($1 \leq i \leq H$) may contain noise caused by other sources, so that the source cardinality may be overestimated. Therefore, we use the minimum value of

estimated source cardinalities $SC_i(s)$ ($1 \leq i \leq H$) as its estimation [40], which is denoted as follows:

$$SC(s) = \min_{i=1}^N SC_i(s). \quad (12)$$

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

$$DC_i(d) = -m_i \ln \frac{U_{B_i(d)}}{m_i}, \quad (13)$$

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

We use the minimum value of estimated destination cardinalities $DC_i(d)$ ($1 \leq i \leq H$) as its estimation, which is denoted as follows:

$$DC(d) = \min_{i=1}^N DC_i(d). \quad (14)$$

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

Input: the updated $B = (B_1, B_2, \dots, B_H)$
Output: supersources and destinations

- (1) **for** $i = 1$ to H **do**
- (2) compute the number $U_{B_i[j][\cdot]}$ of zero bits in each $B_i [j] [\cdot]$
- (3) $RC_i^j \leftarrow -n_i \ln(U_{B_i[j][\cdot]}/n_i)$, $0 \leq j \leq n_i - 1$
- (4) **if** $RC_i^j > \alpha_i$ **then**
- (5) the row $B_i [j] [\cdot]$ is an abnormal row
- (6) **end if**
- (7) compute the number $U_{B_i[\cdot][j]}$ of zero bits in each $B_i [\cdot] [j]$
- (8) $CC_i^j \leftarrow -m_i \ln(U_{B_i[\cdot][j]}/m_i)$, $0 \leq j \leq m_i - 1$
- (9) **if** $CC_i^j > \beta_i$ **then**
- (10) the column $B_i [\cdot] [j]$ is an abnormal column
- (11) **end if**
- (12) **end for**
- (13) Obtain supersources and destinations by inverse calculation

ALGORITHM 3: Detecting supersources and superdestinations.

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

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

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

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

$$d \equiv \sum_{i=1}^H N_i N_i^{-1} c'_i \pmod{N}, \quad (16)$$

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

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

We design a strategy to establish relations between two consecutive rows to which sources are mapped by row hash functions. Taking one source s as example, we obtain the row index $f_i(s)$ ($1 \leq i \leq H$) and the next row index $f_{i+1}(s)$ ($1 \leq i \leq H-1$) using row hash function f_i ($1 \leq i \leq H$). Therefore, the row index $f_i(s)$ is associated with the next row index $f_{i+1}(s)$ ($1 \leq i \leq H-1$) by one hash table, that is, $T_i[f_i(s)] =$

$f_{i+1}(s)$ ($1 \leq i \leq H-1$). However, different sources may be mapped to the same rows, leading to hash collisions. Assuming another source s' , $T_i[f_i(s)] = \{f_{i+1}(s), f_{i+1}(s')\}$ ($1 \leq i \leq H-1$). We conduct row combinations based on the hash table to reduce computational overhead and improve false positive rate and false negative rate. After that, we can accurately recover sources from row combinations using inverse calculation. Similarly, supposing that there are two destinations d and d' mapped to the same columns, $T_i[h_i(d)] = \{h_{i+1}(d), h_{i+1}(d')\}$ ($1 \leq i \leq H-1$). Likewise, we conduct column combinations based on the hash table and recover destinations using inverse calculation to reduce computational overhead and improve false positive rate and false negative rate.

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

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

$$RC_i^j = -n_i \ln \frac{U_{B_i[j][\cdot]}}{n_i}, \quad (17)$$

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

$$CC_i^j = -m_i \ln \frac{U_{B_i[\cdot][j]}}{m_i}, \quad (18)$$

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

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

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

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

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

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

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

$$\ln V_{B_i(s)} = -\frac{SC_i}{n_i} + \frac{V_{B_i(s)} - e^{-SC_i/n_i}}{e^{-SC_i/n_i}} - \frac{1}{2} \cdot \frac{(V_{B_i(s)} - e^{-SC_i/n_i})^2}{e^{-2SC_i/n_i}} + \dots \quad (20)$$

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

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

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

$$E(\widehat{SC}_i) = SC_i + \frac{1}{2} \cdot \frac{n_i E(V_{B_i(s)} - e^{-SC_i/n_i})^2}{e^{-2SC_i/n_i}}. \quad (22)$$

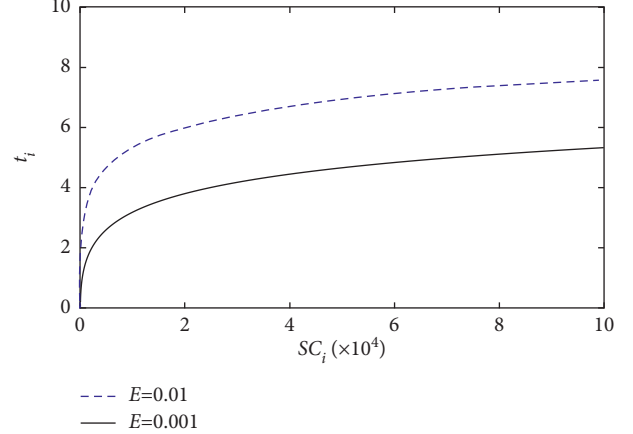


FIGURE 4: The parameters SC_i and t_i impact on the relative error of source cardinality estimation.

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

$$E(\widehat{SC}_i) = SC_i + \frac{e^{SC_i/n_i} - SC_i/n_i - 1}{2}. \quad (23)$$

The mathematical expectation of relative error is denoted as follows:

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

Let $r_i = SC_i/n_i$, $r_i = SC_i/n_i$. Equation (24) is transformed as follows:

$$E\left(\frac{\widehat{SC}_i - SC_i}{SC_i}\right) = \frac{e^{r_i} - r_i - 1}{2 SC_i}. \quad (25)$$

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

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

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

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

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

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

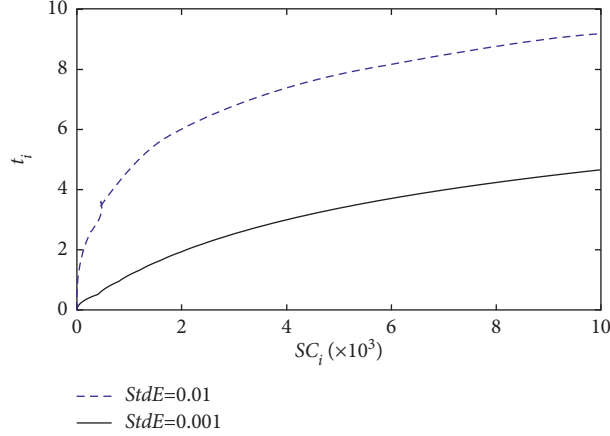
FIGURE 5: The parameters SC_i and t_i impact on the standard error of the ratio \widehat{SC}_i/SC_i .

TABLE 1: Statistics of datasets.

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

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

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

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

Let $r_i = SC_i/n_i$; equation (11) is transformed as follows:

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

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

5. Experiment and Evaluation

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

TABLE 2: True supersources and destinations.

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

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

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

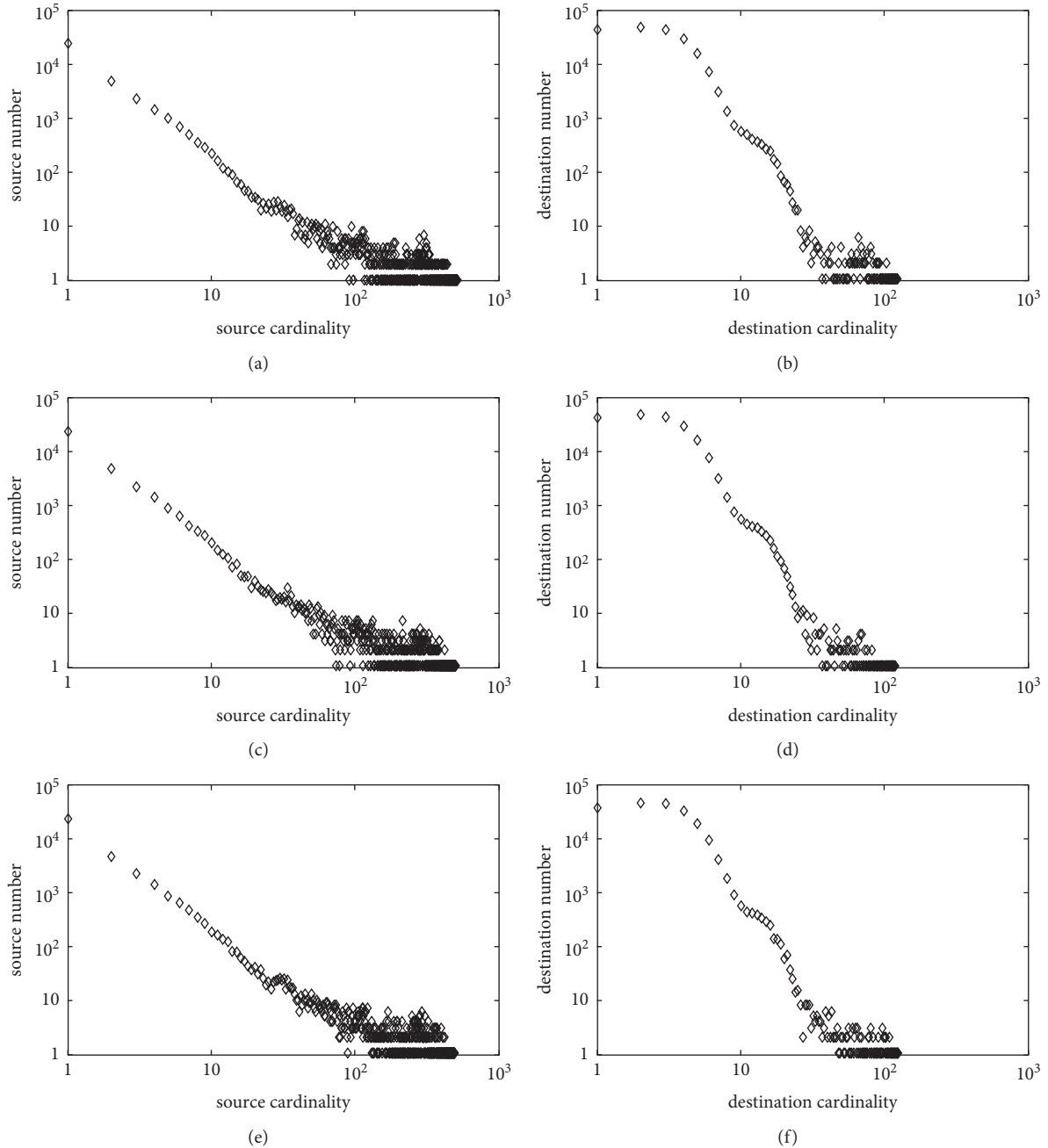


FIGURE 6: Cardinality distribution. (a) Distribution of source cardinality under data1. (b) Distribution of destination cardinality under data1. (c) Distribution of source cardinality under data2. (d) Distribution of destination cardinality under data2. (e) Distribution of source cardinality under data3.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

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

In the experiments, we assume that there are three monitors in distributed monitoring systems, each of which processes the 20-second traffic trace for each traffic trace. For

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

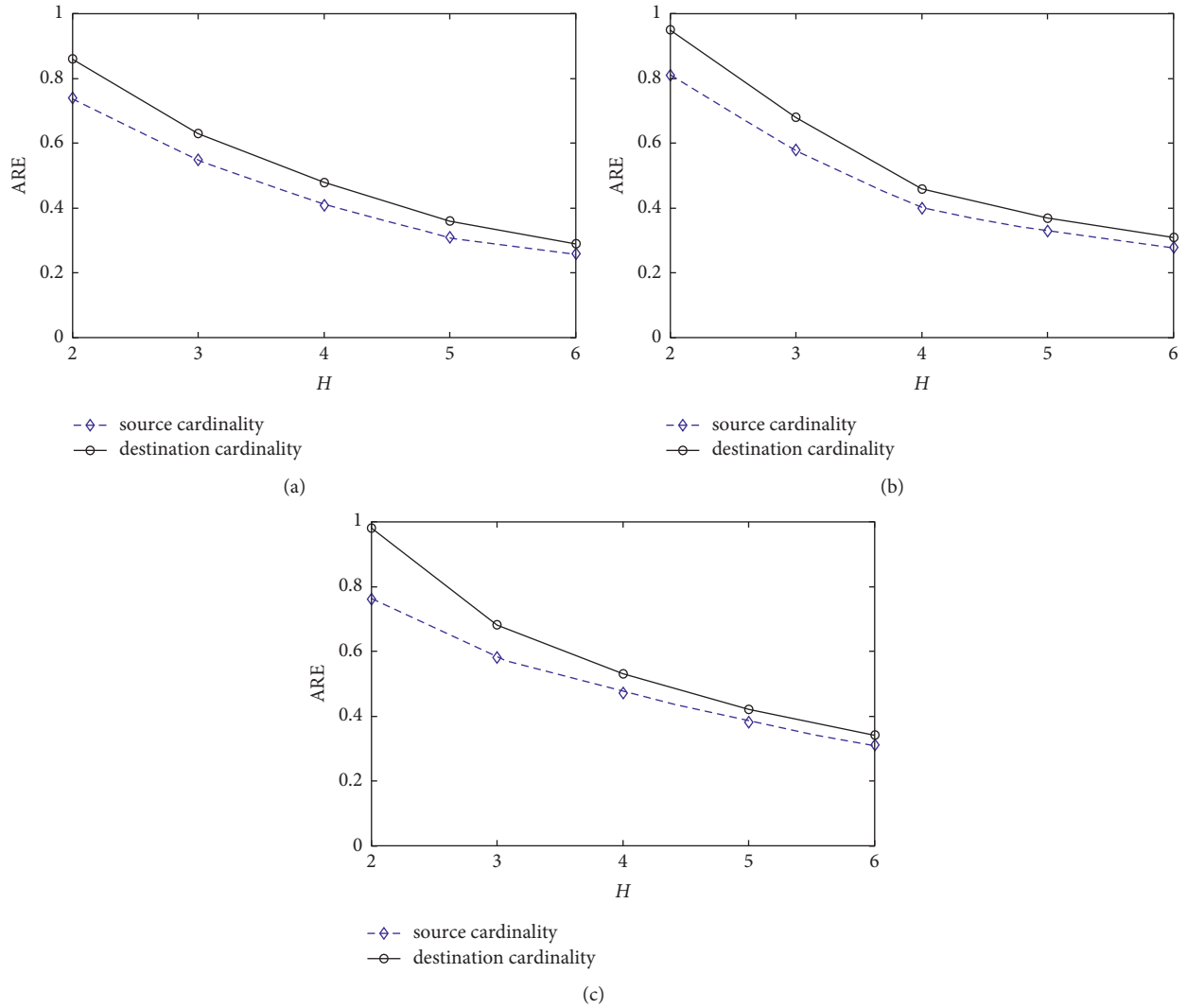


FIGURE 7: The influence of parameter H on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

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

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

5.2. Influence of Parameters on Estimation Accuracy. Our algorithm has three parameters, H , m_r , and n_c , where H denotes the number of hash functions and n_r and m_c denote the number of rows and columns in two-dimensional bit arrays B_i ($1 \leq i \leq H$). Updating, estimating, and reconstructing

processes use hash functions, and the parameter H impacts the processing time. H changes from 2 to 6. Both n_r and m_c determine the size of memory consumed by our algorithm. According to the coupon collection issue, the source and destination cardinality can be accurately estimated when they are less than $n_r \ln n_r$ and $m_c \ln m_c$. In the experiments, source IP addresses and destination IP addresses are used as sources and destinations. Therefore, n_r is a prime from 400 to 800 and m_c is also a prime from 3000 to 7000.

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

$$\text{ARE} = \frac{1}{n} \sum_{i=1}^n \frac{|\hat{c}_i - c_i|}{c_i}, \quad (31)$$

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

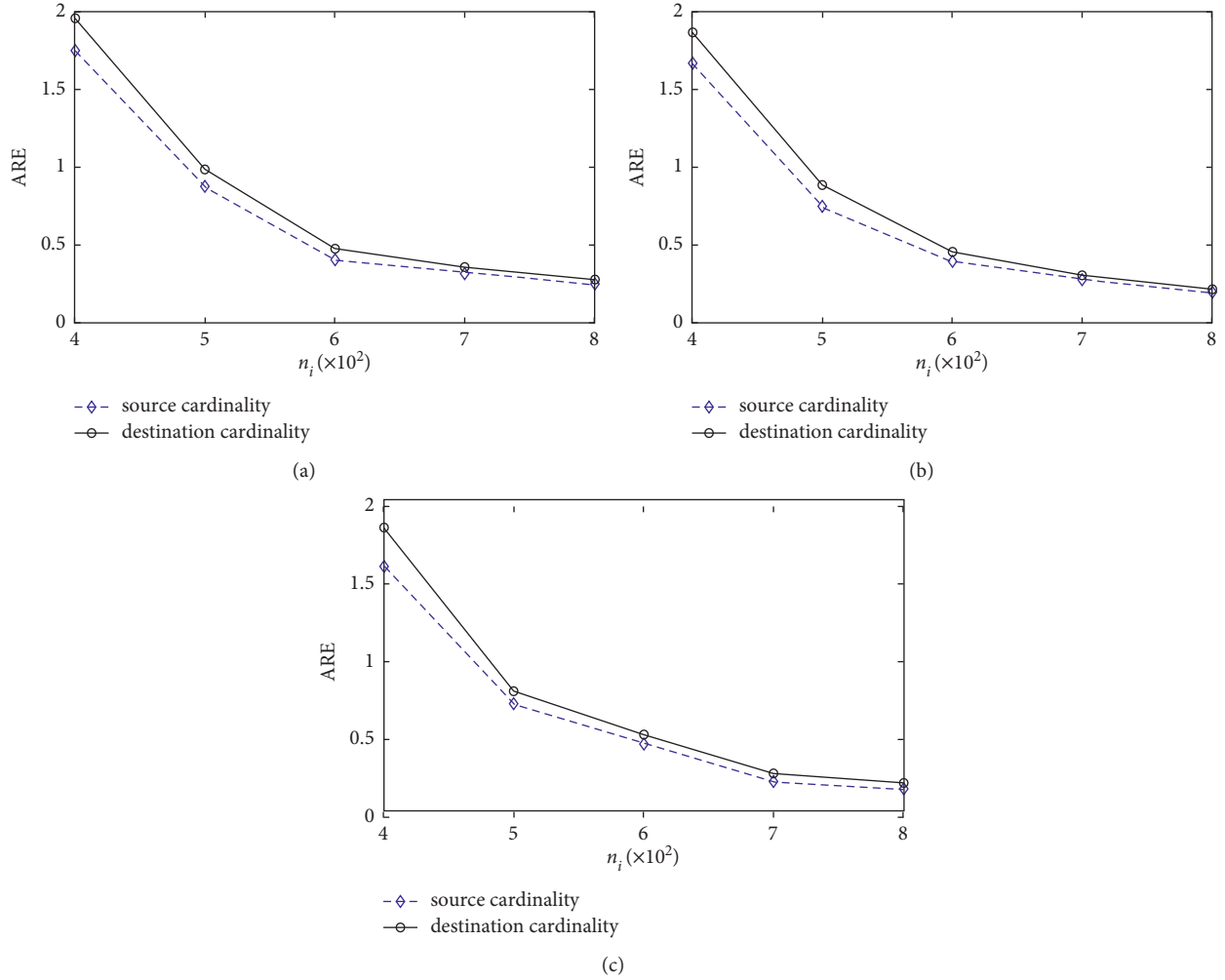


FIGURE 8: The influence of parameter n_i on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

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

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

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

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

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

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

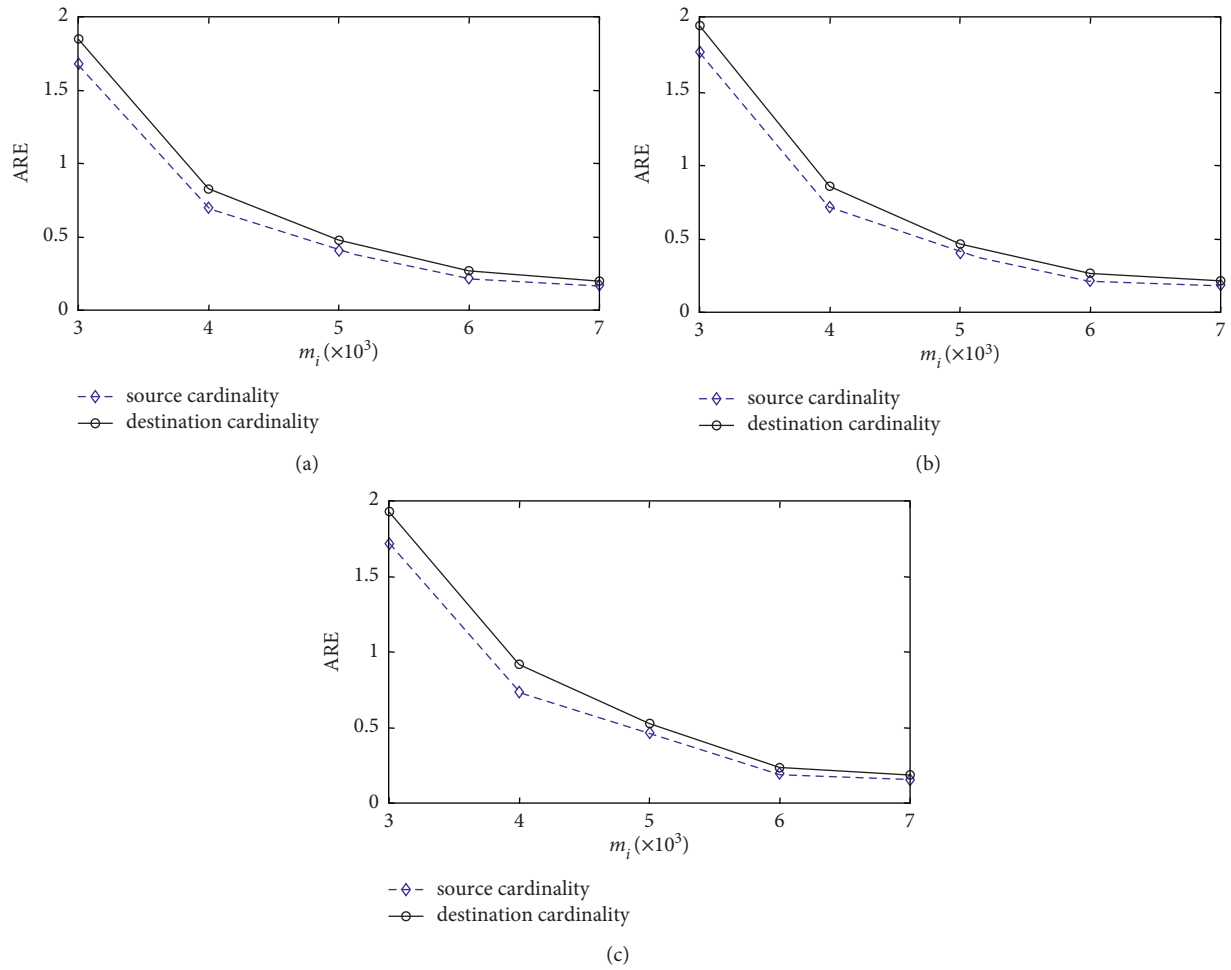


FIGURE 9: The influence of parameter m_i on cardinality estimation under different traffic traces. (a) data1. (b) data2. (c) data3.

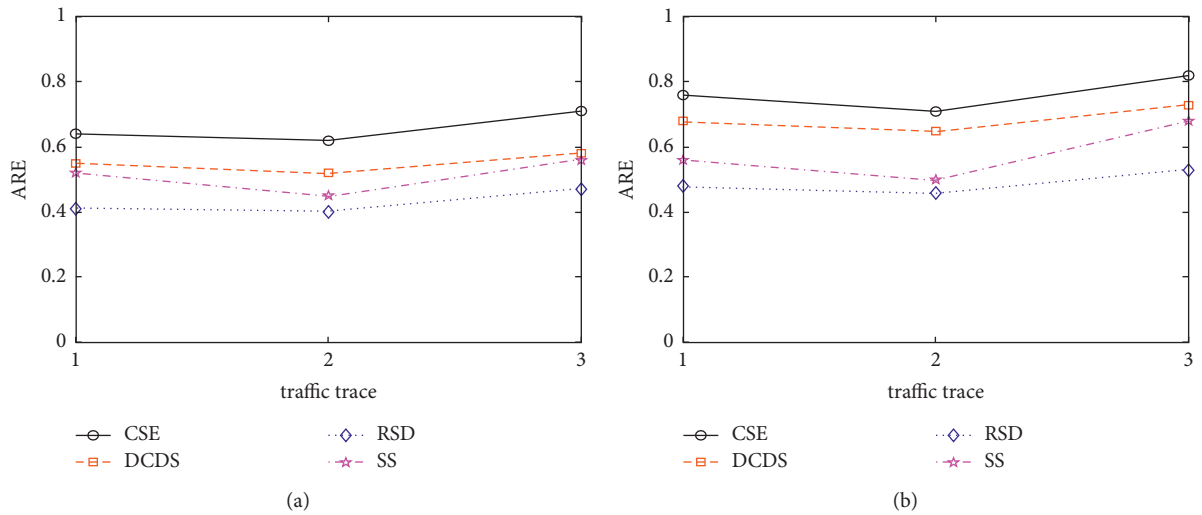


FIGURE 10: Cardinality estimation accuracy. (a) Source cardinality. (b) Destination cardinality.

and accurately estimate the cardinality of sources and destinations based on the same summary data structure generated by the controller in comparison with CSE, DCDS,

and SS. However, the deviation between the estimated cardinality and the theoretical value still exists due to hash collisions.

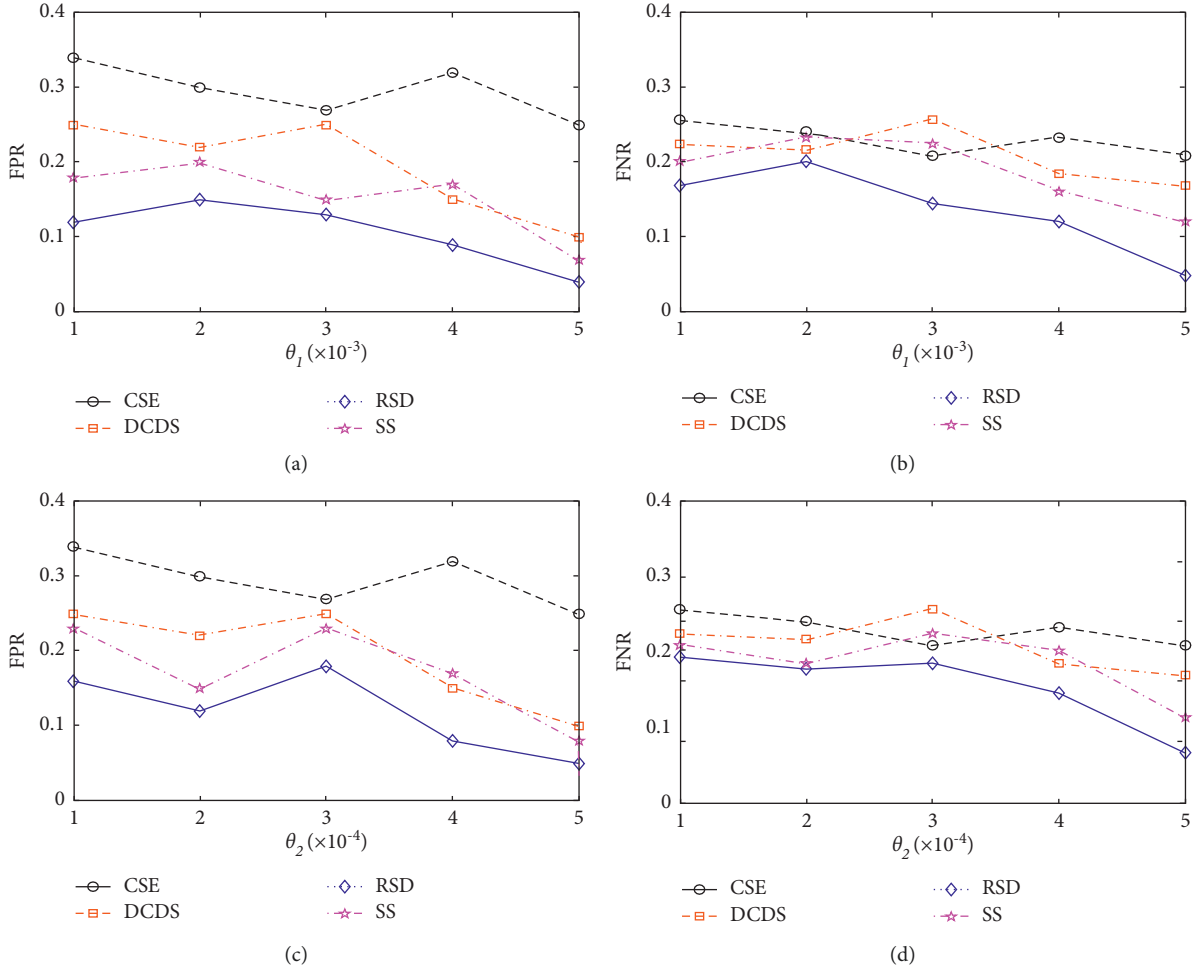


FIGURE 11: Detection precision under data1. (a) False positive rate of supersources. (b) False negative rate of supersources. (c) False positive rate of superdestinations. (d) False negative rate of superdestinations.

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

$$\text{FPR} = \frac{|B - A|}{|B|}, \quad (32)$$

$$\text{FNR} = \frac{|A - B|}{|A|}, \quad (33)$$

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

Figure 11 shows the detection precision of three algorithms under data1. The FPR and FNR of detecting supersources are shown in Figures 11(a) and 11(b). We can see that RSD has the

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

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

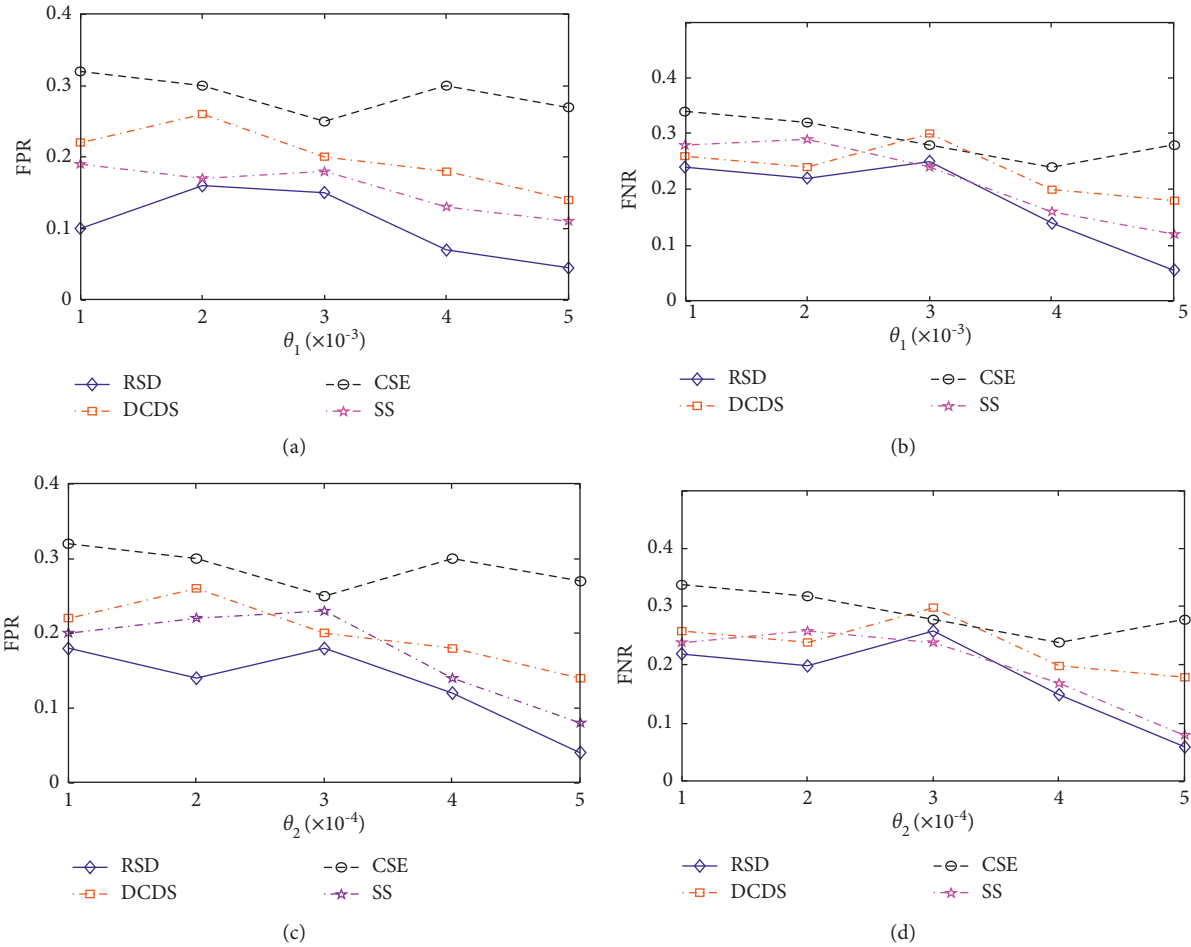


FIGURE 12: Detection precision under data2. (a) False positive rate of supersources. (b) False negative rate of supersources. (c) False positive rate of superdestinations. (d) False negative rate of superdestinations.

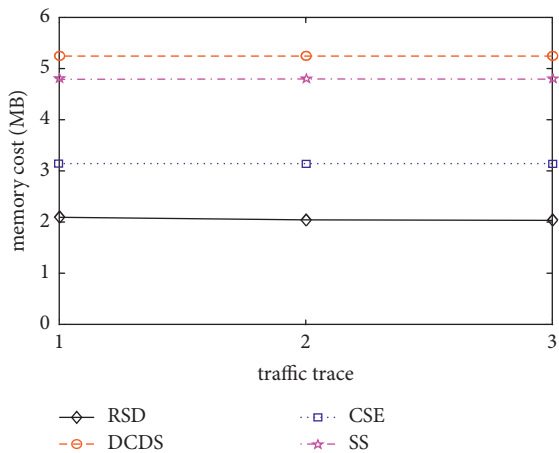


FIGURE 13: Memory cost under different traffic traces.

summary data structure, arrays, and hash table. The memory cost slightly increases when the extra bit arrays and hash table are created in the updating and reconstruction process. For simplicity, we treat the size of summary data structure as the memory cost. CSE, DCDS,

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

6. Conclusion

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

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

Data Availability

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

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

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

References

- [1] T. Yang, H. Zhang, J. Li et al., "HeavyKeeper: an accurate algorithm for finding Top- k elephant flows," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.
- [2] J. Li, Z. Li, Y. Xu et al., "WavingSketch: an unbiased and generic sketch for finding top- k items in data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1574–1584, CA, USA, July 2020.
- [3] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "FlowMon-DPDK: parsimonious per-flow software monitoring at line rate," in *Proceedings of the IEEE Network Traffic Measurement and Analysis Conference*, pp. 1–8, Vienna, Austria, June 2019.
- [4] H. Wang, H. Xu, L. Huang, and Y. Zhai, "Fast and accurate traffic measurement with hierarchical filtering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2360–2374, 2020.
- [5] Y. Du, H. Huang, Y.-E. Sun, S. Chen, and G. Gao, "Self-adaptive sampling for network traffic measurement," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–10, Vancouver, BC, Canada, May 2021.
- [6] Y. Jing, R. Zhang, Y. Ma, Y. Zheng, L. Wu, and D. Zhang, "SuperSketch: a multi-dimensional reversible data structure for super host identification," *IEEE Transactions on Dependable and Secure Computing* 1 page, 2021.
- [7] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 164–176, Santa Clara, CA, USA, April 2017.
- [8] Q. Huang and P. P. C. Lee, "A hybrid local and distributed sketching design for accurate and scalable heavy key detection in network data streams," *Computer Networks*, vol. 91, pp. 298–315, 2015.
- [9] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *Proceedings of the IEEE International Conference on Computer Communications*, pp. 1–9, San Francisco, CA, USA, July 2020.
- [10] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, Atlanta, GA, USA, July 2020.
- [11] L. Tang, Q. Huang, and P. P. C. Lee, "MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 2026–2034, Paris, France, April 2019.
- [12] Q. Xiao, Z. Tang, and S. Chen, "Universal online sketch for tracking heavy hitters and estimating moments of data streams," *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 974–983, Toronto, ON, Canada, July 2020.
- [13] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1172–1185, 2020.
- [14] J. Moraney and D. Raz, "On the practical detection of heavy hitter flows," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pp. 268–276, Bordeaux, France, May 2021.
- [15] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 289–300, 2016.
- [16] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, "Finding persistent items in distributed datasets," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 1–14, 2020.
- [17] L. Chen, H. Dai, L. Meng, and J. Yu, "Finding needles in a hay stream: on persistent item lookup in data streams," *Computer Networks*, vol. 181, Article ID 107518, 2020.
- [18] S. A. Singh and S. Tirthapura, "Monitoring persistent items in the union of distributed streams," *Journal of Parallel and Distributed Computing*, vol. 74, no. 11, pp. 3115–3127, 2014.
- [19] C. Ma, S. Chen, Y. Zhang, Q. Xiao, and O. O. Odegbile, "Super spreader identification using geometric-min filter," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 299–312, 2022.
- [20] P. Wang, X. Guan, T. Qin, and Q. Huang, "A data streaming method for monitoring host connection degrees of high-speed links," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1086–1098, 2011.

- [21] W. Liu, W. Qu, J. Gong, and K. Li, "Detection of superpoints using a vector bloom filter," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 514–527, 2016.
- [22] Y. Liu, W. Chen, and Y. Guan, "Identifying high-cardinality hosts from network-wide traffic measurements," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 547–558, 2016.
- [23] G. Cheng and Y. Tang, "Line speed accurate superspreader identification using dynamic error compensation," *Computer Communications*, vol. 36, no. 13, pp. 1460–1470, 2013.
- [24] J. Wang, W. Liu, L. Zheng, Z. Li, and Z. Liu, "A novel algorithm for detecting superpoints based on reversible virtual bitmaps," *Journal of Information Security and Applications*, vol. 49, Article ID 102403, 2019.
- [25] R. Cohen and Y. Nezri, "Cardinality estimation in a virtualized network device using online machine learning," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2098–2110, 2019.
- [26] H. Harmouch and F. Naumann, "Cardinality estimation," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 499–512, 2017.
- [27] M. Roesch, "Snort: lightweight intrusion detection for networks," in *Proceedings of the USENIX International Conference on Systems Administration*, pp. 229–238, Seattle, WA, USA, June 1999.
- [28] D. Plonka, "FlowScan: a network traffic flow reporting and visualization tool," in *Proceedings of the USENIX International Conference on Systems Administration*, pp. 305–317, New Orleans, LA, USA, December 2000.
- [29] Y. E. Sun, H. Huang, and C. Ma et al., "Online spread estimation with non-duplicate sampling," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 2440–2448, Toronto, ON, Canada, July 2020.
- [30] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual filter for non-duplicate sampling," *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, IEEE, in *Proceedings of the IEEE International Conference on Network Protocols*, pp. 1–11, November 2021.
- [31] L. Zheng, D. Liu, W. Liu, Z. Liu, Z. Liu, and T. Wu, "A data streaming algorithm for detection of superpoints with small memory consumption," *IEEE Communications Letters*, vol. 21, no. 5, pp. 1067–1070, 2017.
- [32] L. Wang, T. Yang, H. Wang et al., "Fine-grained probability counting for cardinality estimation of data streams," *World Wide Web*, vol. 22, no. 5, pp. 2065–2081, 2019.
- [33] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, 2021.
- [34] Y. Zhou, Y. Zhou, S. Chen, and Y. Youlin Zhang, "Per-flow counting for big network data stream over sliding windows," *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, IEEE, in *Proceedings of the IEEE/ACM International Symposium on Quality of Service*, pp. 1–10, June 2017.
- [35] J. Qi, W. Li, T. Yang, D. Li, and H. Li, "Cuckoo counter: a novel framework for accurate per-flow frequency estimation in network measurement," *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, IEEE, in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1–7, September 2019.
- [36] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: accurate per-flow measurement for big streaming data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2650–2662, 2019.
- [37] Y. Zhou, P. Liu, H. Jin, T. Yang, S. Dang, and X. Li, "One memory access sketch: a more accurate and faster sketch for per-flow measurement," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, in *Proceedings of the IEEE Global Communications Conference*, pp. 1–6, Singapore, 4 December 2022.
- [38] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, Honolulu, HI, USA, 23 April 2006.
- [39] R. Shahout, R. Friedman, and D. Adas, "CELL: counter estimation for per-flow traffic in streams and sliding windows," in *Proceedings of the IEEE International Conference on Network Protocols*, pp. 1–12, Dallas, TX, USA, 13 Oct. 2020.
- [40] Q. Liu, H. Dai, A. X. Liu, Q. Li, X. Wang, and J. Zheng, "Cache assisted randomized sharing counters in network measurement," in *Proceedings of the ACM International Conference on Parallel Processing*, pp. 1–10, Eugene, OR, USA, August 2018.
- [41] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.
- [42] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, 2011.
- [43] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: improving the performance of cardinality estimation of large data streams," in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1–9, IEEE, Atlanta, GA, USA, 1 May 2017.
- [44] P. Wang, P. Jia, J. Tao, and X. Guan, "Detecting a variety of long-term stealthy user behaviors on high speed links," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1912–1925, 2019.
- [45] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, 5 pages, 2008.
- [46] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the International Conference on Database Theory*, pp. 398–412, Edinburgh, UK, January 1999.
- [47] F. Wang and L. Gao, "Simple and efficient identification of heavy hitters based on bitcount," in *Proceedings of the IEEE International Conference on High Performance Switching and Routing*, pp. 1–6, Xi'an, China, 6 June 2022.
- [48] H. Huang, Y. E. Sun, C. Ma et al., "Spread estimation with non-duplicate sampling in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2073–2086, 2021.
- [49] L. Tang, Q. Huang, and P. P. C. Lee, "SpreadSketch: toward invertible and network-wide detection of superspreaders," *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, in *Proceedings of the IEEE Conference on Computer Communications*, pp. 1608–1617, Toronto, ON, Canada, 6 July 2020.
- [50] T. Yang, J. Jiang, P. Liu et al., "Elastic sketch: adaptive and fast network-wide measurements," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 561–575, New York, NY, USA, August 2018.
- [51] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide

- measurements,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2386–2398, 2021.
- [52] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, “Persistent spread measurement for big network data based on register intersection,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 1, 67 pages, 2017.
- [53] H. Huang, Y. E. Sun, C. Ma et al., “An efficient k-persistent spread estimator for traffic measurement in high-speed networks,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1463–1476, 2020.
- [54] MAWI Dataset, <https://mawi.wide.ad.jp/mawi/>, 2020.

Research Article

Defending against Deep-Learning-Based Flow Correlation Attacks with Adversarial Examples

Ziwei Zhang  and Dengpan Ye 

Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education,
School of Cyber Science and Engineering, Wuhan University, Wuhan, China

Correspondence should be addressed to Dengpan Ye; yedp@whu.edu.cn

Received 23 September 2021; Revised 8 December 2021; Accepted 23 February 2022; Published 27 March 2022

Academic Editor: Mamoun Alazab

Copyright © 2022 Ziwei Zhang and Dengpan Ye. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Tor is vulnerable to flow correlation attacks, adversaries who can observe the traffic metadata (e.g., packet timing, size, etc.) between client to entry relay and exit relay to the server will deanonymize users by calculating the degree of association. A recent study has shown that deep-learning-based approach called DeepCorr provides a high flow correlation accuracy of over 96%. The escalating threat of this attack requires timely and effective countermeasures. In this paper, we propose a novel defense mechanism that injects dummy packets into flow traces by precomputing adversarial examples, successfully breaks the flow pattern that CNNs model has learned, and achieves a high protection success rate of over 97%. Moreover, our defense only requires 20% bandwidth overhead, which outperforms the state-of-the-art defense. We further consider implementing our defense in the real world. We find that, unlike traditional scenarios, the traffic flows are “fixed” only when they are coming, which means we must know the next packet’s feature. In addition, the websites are not immutable, and the characteristics of the transmitted packets will change irregularly and lead to the inefficiency of adversarial samples. To solve these problems, we design a system to adapt our defense in the real world and further reduce bandwidth overhead.

1. Introduction

Tor is the most popular and low-latency anonymity network that provides anonymous communication services for more than two million people [1]. It includes over 3000 relays that transmit massive encrypt packets and conceal client’s information. Every relay only knows its previous and latter relay’s address.

But flow correlation attacks break this security model. Network-level adversaries, i.e., autonomous systems (ASes) have the power to observe traffic characteristics between client to entry relay and exit relay to the destination server. They can link these data (in particular packet timings and packet sizes) to deanonymize users, as shown in Figure 1. The correlation algorithm used in the beginning studies is usually a traditional method like Pearson correlation or Cosine similarity. Recent research leverages a deep learning model to correlate traffic characteristics with significantly higher accuracies than existing algorithms.

Existing defense methods to detect or mitigate traffic analysis attacks mainly focus on obfuscating encrypt packets, traffic morphing, changing network-level characteristics that does not affect the deep-learning-based attack. And to our best knowledge, existing defenses are all designed to mitigate traffic analysis attacks like website fingerprint attacks or BGP hijack attacks. There is no effective defense faced to flow correlation attack.

Against this strong deep-learning-based attack, the adversarial example is a natural choice for us to confuse CNNs model. So, we explore how effective the adversarial examples defend flow correlation attacks and how to implement defense in the real world.

First, we reconstruct the targeted model that represents state-of-the-art attack and gets the similar accuracy that Milad Nasr et al. [10] mentioned. Second, we evaluate various adversarial example methods’ effects including FGSM, C&W, Deepfool, and BIM. The experimental results show that the success rate of applying adversarial examples

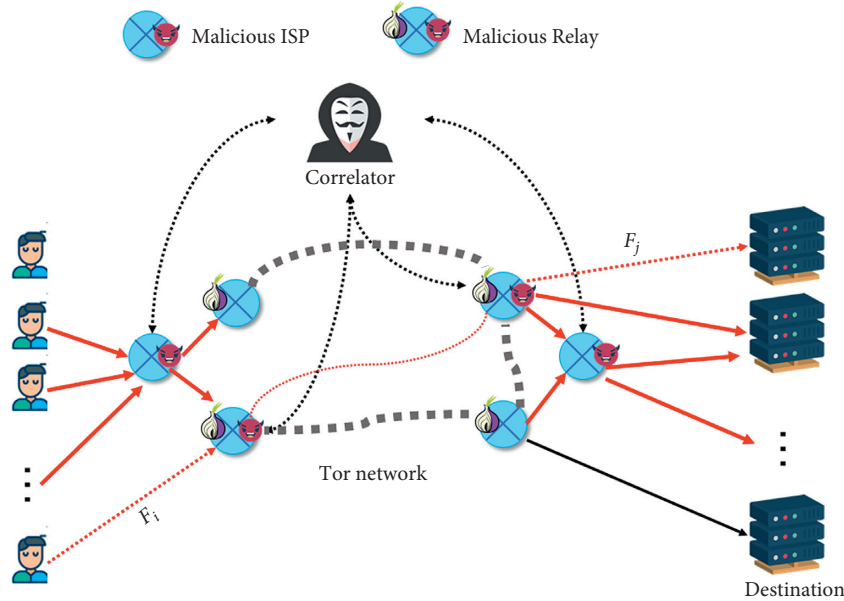


FIGURE 1: The main process of flow correlation attack on Tor. The adversary intercepts Tor flows either by running malicious Tor relays or eavesdropping on Internet ASes and IXPs.

to defeat the flow correlation model is more than 97% with only 20% bandwidth overhead.

Third, we try to implement our defense in the real world, but we find that there are some challenges we have to face. (1) The websites are not immutable, so the characteristics of the transmitted packets are not immutable. (2) The traffic flows are “fixed” only when they are coming, which means we must know the next packet’s feature. (3) The dummy packets we add will go through entire circuit (client- > entry relay- > middle relay- > exit relay- > server). This has increased bandwidth overhead. How can we reduce these extra dummy packets after they have done their job?

To solve the first and second problem, we design a center server that termly collects traffic characteristics of websites and generates corresponding adversarial examples. To solve third problem, we design a mechanism to drop redundant dummy packets at the entry relay, which further reduces bandwidth overhead.

The key contributions of this work are as follows:

- (1) We propose a novel defense mechanism against deep-learning-based flow correlation attacks that inject dummy packets into flow traces by pre-computing adversarial examples.
- (2) We further evaluate various adversarial example methods’ effects, and the experimental results show that even the worst method (FGSM) we used also gain a protection success rate of over 90% with an acceptable bandwidth overhead (30%).
- (3) We analyze the challenges of applying our defense in the real world and design a system to solve these challenges, including center server, full-duplex mode, and drop dummy packets mechanism.

The rest of the paper is organized as follows: Section 2 introduces related work, including the development of traffic

analysis and adversarial examples. Section 3 describes our proposed method in detail. Section 4 shows the details and results of our experiment. In Section 5, we point out our limitations and give future directions. In Section 6, we conclude our work.

2. Related Work

2.1. Flow Correlation Attack and Defense. Flow correlation attack was a type of traffic analysis attack, and the traffic analysis attack was a type of side-channel attack. Side-channel attacks always leveraged non-normal ways to infer sensitive information from well-protected systems, such as by observing traces (e.g., timing, power, or resource usage). Diao et al. [2] launched inference attacks without any permission in Android by interrupting timing analysis and applying it to interrupt logs. Liu et al. [3] presented a side-channel attack to infer user inputs on keyboards by exploiting sensors in the smartwatch. Schuster et al. [4] aimed to identify video information by using the deep-learning model to classify encrypted video streams.

Flow correlation attacks as a significant side-channel attack was applied in many fields. Shmatikov et al. [5] investigated an active attack called watermark attacks. They modified the packet flows to “fingerprint” them and analyze the tradeoffs between the amount of cover traffic, extra latency, etc. In addition, they also proposed a defense method by using adaptive padding. The work of Paxson and Zhang [6] made the traffic packets as a series of ON and OFF patterns and used these data to correlate network flows. Murdoch and Zieliński [8] developed and evaluated Bayesian traffic analysis techniques to process sampled data. Blum et al. [7] correlated the aggregate sizes of network packets over time. Sun et al. [9] further combined the asymmetric traffic analysis and BGP hijacking to deanonymize users.

All the above papers used the static metric standard statistical correlation metrics to correlate the vectors of flow timings and sizes. And to gain a higher accuracy, they need to observe the associated flow for five minutes or more. The time it take was too long to correlate lots of short-lived connections. Nasr et al. [10] were the first one to use CNNs models to learn a flow correlation function and achieve drastically higher accuracies.

There was still a big gap in the defense of flow correlation attacks, and Sun et al. [11] proposed a defense method that mainly solved the BGP hijacking and reduced the chance of adversary observed network traffic. The obfs4 [12] as a Tor official defense could randomly obfuscate packets time and size but get a poor protection success rate with an unacceptable bandwidth overhead. The ScrambleSuit [43] was a thin protocol layer above TCP whose obfuscated the transported application data by using morphing techniques and a secret exchanged out-of-band. It also had impact on defending the flow correlation attacks but has the same problem as obfs4.

There were some ways to improve classification model's ability of defending noisy labels. Liu et al. [47] proved that any surrogate loss function could be used for classification with noisy labels by using importance reweighting. Yu et al. [45] considered the influence of noisy labels in transfer learning and proposed a novel denoising conditional invariant component (DCIC) framework. Xia et al. [46] presented granular-ball sampling that reduced the data size, improved the data quality in noisy label, and get the same classification accuracy on the original data sets. Noise filtering is an effective method of dealing with label noise, but most of them aimed at binary classification. Xia et al. [44] presented a novel label noise filtering learning method for multiclass classification. These methods mainly focus on the scenario of noisy labels that could help adversary improve their correlation model's robustness and our methods aimed at defending against flow correlation attacks by using adversarial examples. Numan et al. [48] carried out a systematic review of clone detection techniques in static WSNs and provided a comprehensive survey of the existing centralized and distributed schemes with their drawbacks and challenges. Guo et al. [49] proposed a deep graph neural network-based Spammer detection (DeG-Spam) model to gain a better effect than baselines that could be a superior choice to correlate with flow data.

2.2. Website Fingerprint Attack and Defense. The scenario and challenge for website fingerprint attacks are very similar to our work. Adversaries get sensitive information about websites such as domain or page content by analyzing network characteristics. It used to be realized by the traditional machine learning method, but now the deep learning method is gradually emerging.

Nowadays more and more studies have been proposed to defeat website fingerprint attacks. Some research focused on the application layer [13–15], defenders changed the routing algorithm or confused HTTP requests to make adversary touch real traffic as little as possible. Application-layer

defense strategies were often difficult to implement because the premise of their implementation was very harsh, such as target websites only had HTTP protocols. And these methods could not defend deep-learning-based attacks (less than 60% protection success rate). Other researches focused on the network layer. They aimed to fool the classification model by inserting dummy packets. In the earlier studies [16–18], they used constant rate padding to reduce information leakage caused by time intervals and traffic volume. However, these methods always require high bandwidth overhead of 150%. A recent study [19] found that inserting packages between two packets with a large time gap would reduce the bandwidth overhead. They were also useless when applied in defending deep-learning-based attacks (only achieve 9% and 28% protection success rate). Finally, there was a super sequence defense method called Walkie-talkie [20], which committed to finding a longer package trace that contains subsequences of different website traces. But it only gets a 50% protection success rate against DNN attacks. In general, no method can maintain a high success rate with a small amount of bandwidth overhead. All related works are present in Table 1.

2.3. Adversarial Examples. Adversarial examples are a series of methods to fool machine learning models, such as deep neural networks. They add perturbations to the clean input, forward it to the classifier and get an unexpected result. How to generate adversarial perturbations becomes a hot topic in computer vision, natural language processing, etc. There were many prior works that had shown first-order gradient-based attacks to be fairly effective in fooling DNN-based models in both image [21–27], audio [28–30], and text [31–33] domains. The idea of such adversarial attacks was to find a good trajectory that maximally changed the value of the model's output and pushed the sample towards a low-density region. However, to our best knowledge, there is no paper to apply adversarial examples in defending deep-learning-based flow correlation attacks.

3. Method

In this section, we introduce the target model and the specific details of defending against deep-learning-based flow correlation attacks with adversarial examples. Next, we will show our system that were designed to implement defense in the real world.

3.1. Target Model. We reconstruct the idea of Milad Nasr et al. to perform traffic correlation attacks. They use a convolutional neural network (CNN) model to learn a correlation function for Tor's noisy network. It is composed of two convolutional layers and three fully connected layers. The input is a flow pair called $F_{i,j}$, which represents two bidirectional network flows i and j . The specific of $F_{i,j}$ can be described as follows:

$$F_{i,j} = [T_i^u, T_j^u, T_i^d, T_j^d, S_i^u, S_j^u, S_i^d, S_j^d], \quad (1)$$

TABLE 1: The related work of flow correlation attack, flow correlation defense, and website fingerprint defense.

Scheme	Method	Innovation points	Authors	Drawbacks
Flow correlation attack	Watermark attacks	Modified the packet flows to “fingerprint” them.	Shmatikov et al. [5]	Require high privileges and break the original communication easily.
	Timing based	Use the traffic patterns to correlate flows.	Paxson and Zhang [6]	Low accuracy.
	Bayesian traffic analysis	Developed Bayesian traffic analysis techniques to process sampled data.	Murdoch and Zieliński [8]	Cannot correlate lots of short-lived connections.
	Fine-grained level detection	Correlated the aggregate sizes of network packets over time.	Blum et al. [7]	Low accuracy.
	Asymmetric traffic analysis	Further combined the asymmetric traffic analysis and BGP hijacking to deanonymize users	Sun et al. [9]	Only useful for BGP hijacking.
	Deep learning based	Use CNNs models to learn a flow correlation function and achieve drastically higher accuracies.	Nasr et al. [10]	Require hardware support.
Flow correlation defense	Counter-RAPTOR	Reduced the chance of adversary observed network traffic.	Sun et al. [11]	Only useful for defending BGP hijacking.
	Obfs4	Randomly obfuscate packets time and size.	Tor project. [12]	Unacceptable bandwidth overhead.
	ScrambleSuit	Use morphing techniques.	Winter et al. [43]	Unacceptable bandwidth overhead.
Website fingerprint defense	Application layer defense	Changed the routing algorithm or confused HTTP requests.	Wladimir et al. [13] Giovanni et al. [14] Henri et al. [15]	Hard to implement in real world.
	Network layer defense	Fool the classification model by inserting dummy packets.	Juarez et al. [19] Wang et al. [20]	Cannot defend the deep-learning-based attack.

where T is the vector of interpacket delays, S is the vector of packet size, and u and d stand for “upstream” and “downstream,” respectively (e.g., S_i^u represents the upstream packet size of i).

The model hyperparameters we choose are consistent with Milad Nasr, which are presented in Table 2. To take a first look at the performance, we train our model using data set that publishes with the paper [10]. It includes 50,000 pairs of associated flow pairs and $50,000 \times 24,999 \approx 1.24 \times 10^9$ pairs of nonassociated flow pairs. And we gain a similar performance as described in the paper.

DeepCorr is able to achieve such high accuracy using only 300 packets of each flow. It tells us that we must take action to prevent AS/ISP level adversaries from compromising the anonymity and privacy of Tor users. In the next chapter, we will introduce the defense effect of the adversarial sample against flow correlation attack model and the system we designed to make the defense method applicable to the real world.

3.2. Adversarial Samples against Flow Correlation Attack.

Due to the popularity of artificial intelligence and deep learning, adversarial samples have appeared in various scenarios and practical applications. But in many cases, adversarial samples are usually used as a means of attack to escape detection models. In our experiments, adversarial samples are used as a means of defense to fight adversaries who eavesdropping or analyzing users’ traffic. Therefore, our defense strategies focus on improving the protection success rate, every small increase will have a huge impact on the

TABLE 2: The model hyperparameters of target model.

Layer	Details
Convolution layer 1	Kernel num: 2000
	Kernel size: (2, 30)
	Stride: (2, 1)
Max pool 1	Activation: Relu
	Window size: (1, 5)
Convolution layer 2	Stride: (1, 1)
	Kernel num: 1000
	Kernel size: (2; 10)
Max pool 2	Stride: (4, 1)
	Activation: Relu
Fully connected 1	Window size: (1, 5)
	Stride: (1, 1)
	Size: 3000, activation: Relu
Fully connected 2	Size: 800, activation: Relu
Fully connected 3	Size: 100, activation: Relu

adversaries. Because traffic flows are very large, the adversary who eavesdropping traffic will take a lot of manual analysis time if the attack success rate cannot reach 95%, which almost means that this method of attack is no longer meaningful. This is the first difference between applying adversarial samples in defending flow correlation attacks and other traditional fields. In addition, every clean image or text is “fixed” before adding perturbation. However, traffic flows will be “fixed” only when it’s coming. That means we must know the next packet’s feature and add corresponding adversarial perturbation. This is the second difference

between applying adversarial sample in defending flow correlation attacks and other traditional fields.

To generate adversarial example, we use different methods: FGSM [34], C&W [36], Deepfool [37], and BIM [35]. The reason for choosing these four methods is to get a more comprehensive evaluation including gradient-based methods and optimization-based methods.

The fast gradient sign method (FGSM) was proposed by Goodfellow et al. in 2015. This algorithm performs a single gradient ascent step as the following formula:

$$\mathbf{x}^* = \mathbf{x} + \eta \text{sign}(\nabla_x L(g(\mathbf{x};\theta), y)), \quad (2)$$

\mathbf{x} is the original input sample, $g(\mathbf{x};\theta)$ presents the model parameterized by θ , y is the label corresponding to the x , and the $L(g(\mathbf{x};\theta), y)$ is the loss function of the classifier. ∇_x is the gradient of the given loss L , which means the direction where the loss increases the most.

We can control bandwidth overhead from small to large by adjusting param η .

Optimization-based attack C&W was proposed by Carlini & Wagner in 2017. This algorithm generates adversarial perturbation based on certain constraints as the following formula:

$$\min \|\delta\|_p^2 \text{ s.t. } g(\mathbf{x} + \delta) \neq y \text{ and } \mathbf{x} + \delta \in X, \quad (3)$$

\mathbf{x} is also the original input sample and the added perturbation is constrained by \mathcal{L}_p to keep small. The $g(\mathbf{x} + \delta)$ is the obtained result under constraint conditions.

The basic iterative method (BIM) was proposed by A. Kurakin in 2016, it increases the loss of the classifier by adjusting the direction after each step. It iteratively computes as following:

$$\mathbf{I}_p^{i+1} = \text{Clip}_\epsilon \{ \mathbf{I}_p^i + \alpha \text{sign}(\nabla J(\theta, \mathbf{I}_p^i, \ell)) \}, \quad (4)$$

\mathbf{I}_p^i presents the perturbed input at the i^{th} iteration and $\text{Clip}_\epsilon\{\cdot\}$ clips the input in its argument at ϵ and α determines the step size. The BIM algorithm starts with $\mathbf{I}_p^0 = \mathbf{I}_c$ and runs for the number of iterations determined by the formula $\lfloor \min(\epsilon + 4, 1.25\epsilon) \rfloor$.

Deepfool was proposed by Moosavi-Dezfooli, it perturbs the input by a small vector, which is computed to take the resulting image to the boundary of the polyhedron at each iteration. The final perturbation is accumulated by perturbations added in each iteration when the original decision boundaries of the network change their label.

All these adversarial sample methods are designed to add perturbations to the area of the entire image. But in our scene, we can only change the traffic characteristics between client and entry relay. So, we can only change the part of matrix data. In addition, the ways we add perturbations are by padding packet to change packet size and inserting dummy packets to change interpacket delays. Thus, the value of our adversarial perturbation will always be positive. In order to achieve these requirements, we add extra constraints as follows:

$$\text{St. } \begin{cases} P > 0, \\ x \in S, S = \{T_i^u, T_i^d, S_i^u, S_i^d\}, \end{cases} \quad (5)$$

where P presents the perturbations value we add, x presents the input, and S presents the area we can change.

3.3. Implement Defense in the Real World. When we think about the actual implementation of our defense in the real world, we must face other challenges. First, the websites are not immutable, and the manager could deploy new functionality, update index pages, launch new activities, etc. So, the characteristics of the transmitted packets will change irregularly and lead to the inefficiency of adversarial samples. Second, we have talked about the limit of adding perturbation in Section 3.2, and we know that only traffic between client and entry relay can be changed. Under this circumstance, we will consider two modes naturally: full-duplex and simplex, who is better? Third, due to network fluctuations, packets might be delayed or received quicker, which will cause the precomputing adversarial examples loss its effect. To meet these requirements, we design a system consisting of some components, as shown in Figure 2.

To solve the first problem, we create ‘‘traffic consensus’’ concept that derives from Tor consensus [38] and stores in a center server. Users can fetch this traffic consensus before connecting to the destination server and add perturbation into live traffic according to the content of traffic consensus. In this traffic consensus, we build the mapping relationship that has the key of website domain and value of corresponding traffic characteristics.

There is an automatic crawler system that collects traffic characteristics termly behind this traffic consensus. Our center server has a Tor client that will request W_n websites that users mostly access like Alexa top 50,00 every T time and check the live status by status code. In addition, we made our exit Tor traffic tunnel through our own SOCKS proxy server. Thus, we can capture ingress Tor flows and the egress Tor flows. If the monitored website is live, dump the traffic file p by tcpdump. Next, we will process p and extract traffic characteristics including the first 300 packets’ size and delays. Finally, we will use these data to generate adversarial samples and write them to the traffic consensus with the website domain. The specific details are shown as Algorithm 1.

To solve the second problem, we need to consider differences between full-duplex and simplex. The simplex means that only inserting dummy packets into flows from client to entry relay, it could be done more easily because we can add perturbation at Tor client directly. But it brings other problems: the area where we can add perturbation is further limited and bandwidth overhead is too large to bear.

The full-duplex means we can add perturbation form client to entry relay and entry relay to client. It has more area to add perturbation than simplex. However, the dummy packets we add will go through entire circuit (client- > entry relay- > middle relay- > exit relay- > server). This has definitely increased bandwidth overhead. Thus, we design a drop dummy packets mechanism to further reduce bandwidth overhead. The goal of our approach is to letting adversaries to eavesdrop on dummy packets, and the circuit does not pass through dummy packets. Therefore, we should have a

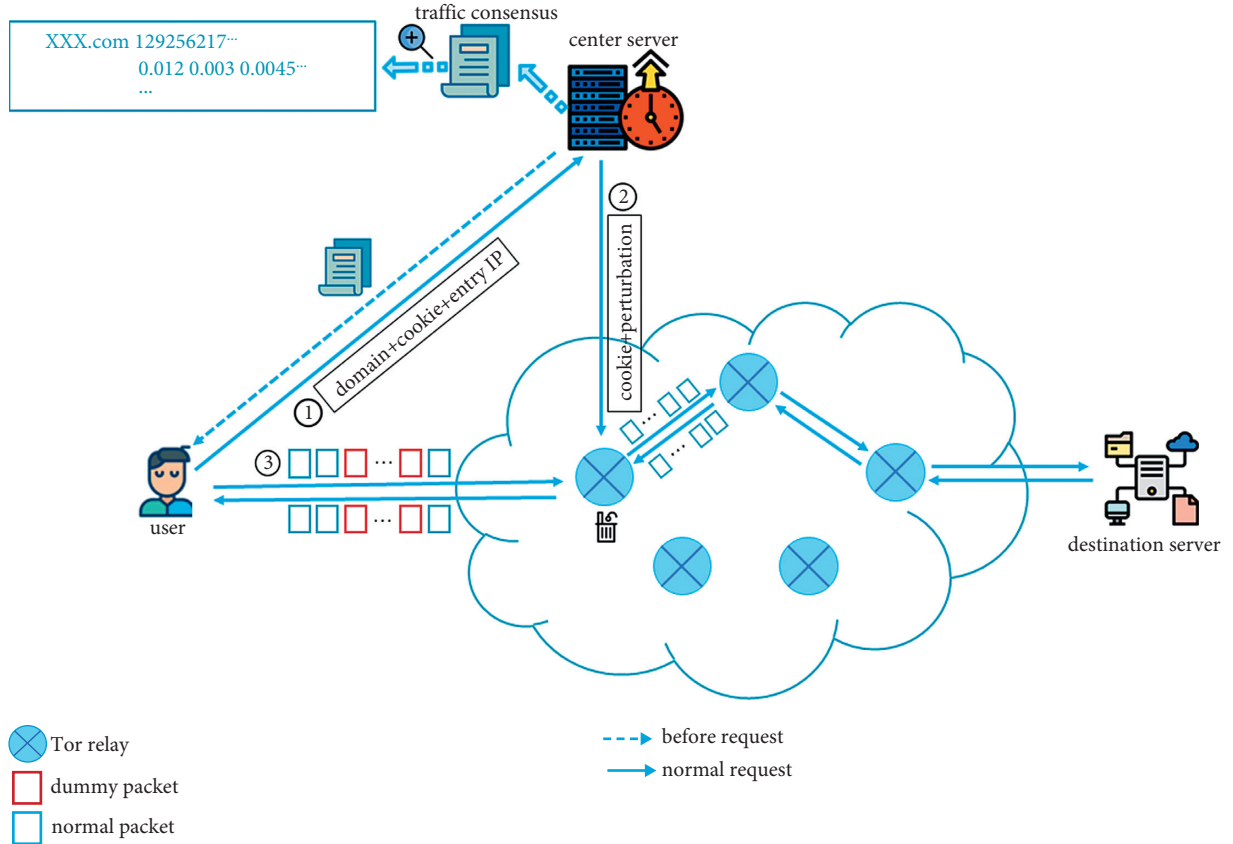
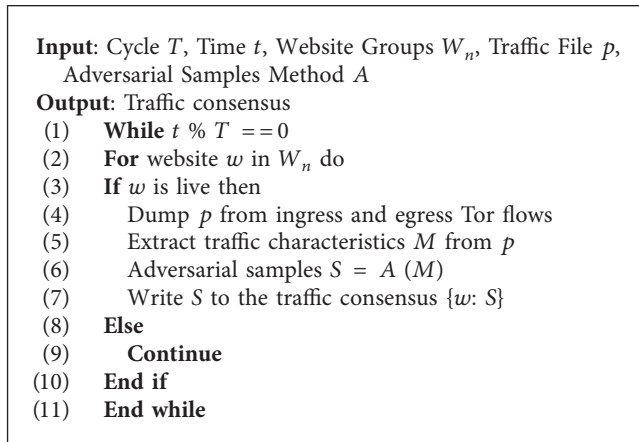


FIGURE 2: The overview of our system and the entire process of protecting users to request websites from flow correlation attacks.



ALGORITHM 1: Traffic consensus generate algorithm.

reasonable method to drop dummy packets at the entry relay. We introduce a new control cell INFO, which is referred to in this paper [13]. This cell will record the order of transmitted packets and be send to entry relay before communication begins. Once the entry relay receives INFO cell, it will drop the extra dummy packets that we add at Tor client according to the cell INFO information and send the packets that truly participate in communication to the middle relay.

In addition, the circuit from entry to the client is controlled by entry relay, which means adding

perturbation is finished by entry relay. We must provide anonymity that entry relay should not know about users' information of visiting websites. To achieve this goal, our idea is inspired by the rendezvous cookie applied in Tor onion services [39] to establish a connection between the user and an onion service. The users will send a cell that consists of the website domain, which will be visited, a cookie that is a 20-byte cryptographic nonce chosen randomly by the users, and the entry relay's IP to the center server. Once center server receives this cell, it will send the perturbation according to the traffic consensus and the cookie generated by users to the entry relay. The entry will store this cookie and perturbation. When users begin to connect to the entry relay with the cookie, the entry relay will compare the cookie that it stores with the cookie that users send. If they are the same, the entry relay will add perturbation into flows to the client. For the third problem, because we already have the drop dummy packets mechanism and full-duplex mode, the only thing what we must do is buffering subsequent cells until the missing cell arrives at the entry relay.

Implementation: we did not implement all components, because it is a large project that needs the entire Tor community's help to modify Tor source code. But we have designed a set of plans as mentioned above and done a lot of experiments to prove the feasibility of our defense including various adversarial samples methods' effect, traffic consensus used time, the advantage of full-duplex brings less

bandwidth overhead, etc. We will show the results in detail in the next section.

4. Results

In this section, we perform a systematic evaluation of our work. Specifically, we compare various adversarial example methods' effects and efficiency against the flow correlation attack model. In our system, we have talked about the advantage and challenges that full-duplex brings, we will further show that our methods' high performance. In addition, we will compare our defense with the state-of-the-art method, and we will test our defense against the traditional flow correlation attack method.

4.1. Data Set. Tor Flow Correlation Data set: in our experiments, we choose to use the public data set of DeepCorr [40]. This data set contains a large number of Tor flows that are captured by visiting Top Alexa's websites. The storage form in the data set is pickle file, which contains the packet size and interpacket delays. Meanwhile, flow pair that belongs to the same Tor connection (associated flow) is labeled with 1, and the flow pair that belongs to arbitrary Tor connections (nonassociated flow) is labeled with 0. We evaluate our system's performance with 9000 flows.

Sirinam and Rimmer Data set: to our best knowledge, the public flow correlation attack data set has only one that is released by Nasr et al. [10]. But we have pointed our scenarios and challenges are very similar to website fingerprint attacks. Thus, we use two well-known WF data sets including Sirinam et al. [41] and Rimmer et al. [42] to evaluate our system's performance. They both contain Tor users' flow pairs and their corresponding websites. The specific details of these two data sets are presented in Table 3.

4.2. Experiment Results. We test FGSM, C&W, Deepfool, and BIM on the same test data set that contains 9000 flows and compares their protection success rate with the same bandwidth overhead (all use the L_∞ perturbation norm). Except for DeepCorr flow correlation attack, we also test our defense against traditional flow correlation attacks including RAPTOR, Pearson, and Cosine. Table 4 shows the result, and we can see even the worst method FGSM could get the 71.2% protection success rate with only 25% bandwidth overhead, and it is also effective against traditional flow correlation attacks. We must point that because the Pearson and Cosine methods use the static metric to measure the correlation, any slight perturbation will have a big impact on the result. Even our method is oriented to the deep-learning-based attack, and the perturbation we added will also break the pattern that the Pearson and Cosine catch.

We also evaluate FGSM, C&W, Deepfool, BIM against website fingerprint attacks including deep-learning-based attack Var-CNN and non-DNN attacks k-NN, k-FP on the Sirinam, Rimmer data set. Table 5 shows the protection success rate of our method, and we can find adversarial examples is effective for defending WF attacks that get sensitive information by classification model.

TABLE 3: Two WF data sets used by our experiments.

Data set name	Labels	Training flows (K)	Testing flows (K)
Sirinam	95	7	1
Rimmer	900	5	0.8

In Chapter 3.3, we have talked about the difference between full-duplex and simplex. Full-duplex has more area to add perturbation and less bandwidth overhead because of dropping dummy packets mechanism. Figure 3 shows that how effective of two modes are with FGSM. We find full-duplex mode has a higher protection success rate than simplex mode with the same bandwidth overhead and the same adversarial example generation method. In addition, our system will update the traffic consensus termly, which means that this process must be within a tolerable time frame. We evaluate our system's efficiency on a PC computer that has an i7 11700k CPU and four GTX 2080Ti GPU. We evaluate the total time of generating 500 websites' traffic consensus and adversarial examples on our test data set. Table 6 shows the result, and we can see that our system is very portable. The FGSM method can update the adversarial perturbation in 1575 s seconds. And we should be aware that our hardware is limited, and anyone can extend the hardware environment to further reduce time consumption.

In Table 4, we can see the FGSM gets the worst protection success rate compared to other methods. But because it is a one-step method, it has the highest efficiency. In our system, time consumption is an important indicator because when the website we focus on become more and more, small-time consumption will be magnified a zillion time over. As for the protection success rate, FGSM get 71.2% with 20% bandwidth overhead. It looks a little low, but as we all know traffic flows are very large, adversaries who eavesdropping traffic will take a lot of manual analysis time if the attack success rate cannot reach 95%, and it almost means that this attack is no longer meaningful. Figure 4 shows the protection success rate of FGSM as bandwidth overhead changes, and we can see it also can get a 95% protection success rate with 35% bandwidth overhead, which is lower than state-of-art defense.

4.3. Comparison

4.3.1. Obfs4. To our best knowledge, obfs4 is the state-of-art and official defense. It is a Tor's pluggable transports to defeat censorship by nation-states who block all Tor traffic. obfs4 modified packet timings and packet sizes to defeat flow correlation, by padding or splitting packets, or by delaying packets to perturb their timing characteristics. Table 7 shows that our defense protection success rate compares with obfs4. Table 8 shows that our defense bandwidth overhead compares with obfs4. Our defense has advantages both in protection success rate and bandwidth overhead.

4.3.2. ScrambleSuit. ScrambleSuit [43] is a thin protocol layer above TCP whose obfuscates the transported application data by using morphing techniques and a secret

TABLE 4: The protection success rate of various adversarial examples against flow correlation attacks with the same bandwidth overhead. PSR presents the protection success rate.

Method	Bandwidth overhead (L_{∞})	DeepCorr (PSR) (%)	RAPTOR (PSR) (%)	Pearson (PSR) (%)	Cosine (PSR) (%)
FGSM	0.20	71.2	93.8	97.5	97.2
C&W	0.20	97.4	93.5	97.2	96.4
Deepfool	0.20	93.6	92.7	96.9	96.2
BIM	0.20	87.5	95.6	97.2	95.8

TABLE 5: The protection success rate of various adversarial examples against website fingerprint attacks with the same bandwidth overhead. PSR presents the protection success rate.

Data set	Method	Bandwidth overhead (L_{∞})	K-NN (PSR) (%)	K-FP (PSR) (%)	Var-CNN (PSR) (%)
Sirinam	FGSM	0.20	97.2	97.4	80.3
	C&W	0.20	99.4	99.5	88.8
	Deepfool	0.20	99.7	99.3	94.5
	BIM	0.20	98.5	98.8	86.7
Rimmer	FGSM	0.20	97.2	96.7	86.7
	C&W	0.20	99.9	99.3	93.2
	Deepfool	0.20	98.7	98.1	97.5
	BIM	0.20	98.2	97.5	89.4

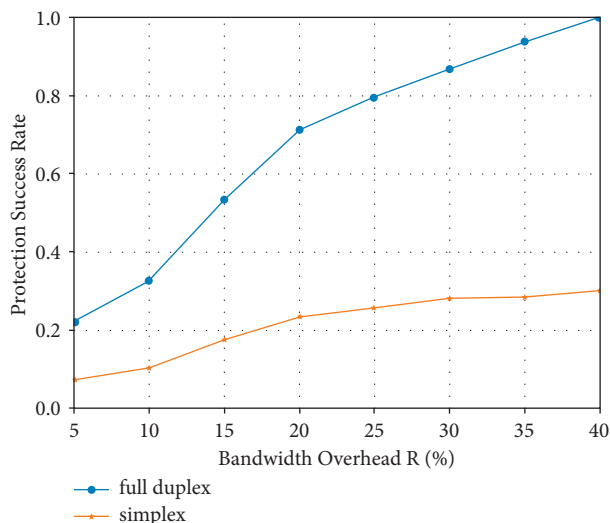


FIGURE 3: Compare full-duplex with simplex mode under the same conditions.

TABLE 6: Our system’s time consumption under the limited hardware environment.

	Traffic consensus	FGSM	C&W	Deepfool	BIM
Time (s)	1205	1575	47382	28377	4858

exchanged out-of-band. It also has impact on defending the flow correlation attacks. Table 7 shows that our defense protection success rate compares with ScrambleSuit. Table 8 shows that our defense bandwidth overhead compares with ScrambleSuit.

4.3.3. *Blind Adversary.* Blind Adversary [50] create universal adversarial perturbations by GANs (generative adversarial networks). This approach protects against both flow

correlation attack and website fingerprint attack but require significant additional resources and bandwidth overhead. Table 7 shows that our defense protection success rate compares with Blind Adversary. Table 8 shows that our defense bandwidth overhead compares with Blind Adversary.

5. Limitations and Future Directions

As mentioned earlier, this work is focused on defeating CNN-based flow correlation attacks with adversarial

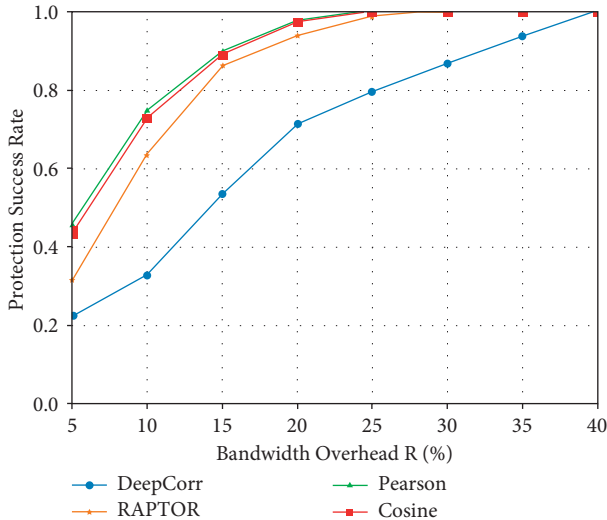


FIGURE 4: The protection success rate of FGSM as bandwidth overhead changes.

TABLE 7: The protection success rate of our defense, obfs4 and ScrambleSuit with the same bandwidth overhead.

Method	Bandwidth overhead (L_{∞})	DeepCorr (PSR) (%)
FGSM	0.20	71.2
C&W	0.20	97.4
Deepfool	0.20	93.6
BIM	0.20	87.5
Obfs4	0.20	44.5
ScrambleSuit	0.20	22.8
Blind adversary	0.20	88.9

TABLE 8: The bandwidth overhead of our defense, obfs4 and ScrambleSuit with the same protection success rate.

Method	Bandwidth overhead (L_{∞})	DeepCorr (PSR) (%)
FGSM	0.41	100
C&W	0.22	100
Deepfool	0.24	100
BIM	0.27	100
Obfs4	0.78	100
ScrambleSuit	0.82	100
Blind adversary	0.33	100

examples. At present, there are a lot of research about defending the adversarial examples, and the adversarial training is one of the most effective approaches. Adversary can compute our adversarial perturbations and retrain their models against them to improve robustness. Future work can extend our system to defeat adversarial training and other methods that aim to reduce the effect of adversarial examples.

6. Conclusion

In this paper, we evaluate the effect of using adversarial samples to defend flow correlation attacks, and the

experimental results show that we achieved a good performance. We further consider implementing our defense in the real world. And we find some challenges we must face. To solve these problems, we design a system including traffic consensus, full-duplex mode, and drop dummy packets mechanism. Our system not only makes adding adversarial perturbations become reality but also further reduce bandwidth overhead.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China NSFC (grant nos. 62072343 and U1736211) and the National Key Research Development Program of China (grant no. 2019QY(Y)0206).

References

- [1] T. Metrics, “Tor Metrics,” 2021, <https://metrics.torproject.org/userstats-relay-country.html>.
- [2] W. Diao, X. Liu, Li Zhou, and K. Zhang, “No pardon for the interruption: new inference attacks on android through interrupt timing analysis,” in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, May, 2016.
- [3] X. Liu, Z. Zhou, and W. Diao, “When good becomes evil: keystroke inference with smartwatch,” in *Proceedings of the CCS’15: 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, Colorado, US, October, 2015.
- [4] R. Schuster, V. Shmatikov, and E. Tromer, “Beauty and the burst: remote identification of encrypted video streams,” *26th USENIX Security Symposium (USENIX Security)*, vol. 17, 2017.
- [5] V. Shmatikov and M.-H. Wang, “Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses,” in *Proceedings of the European Symposium on Research in Computer Security*, pp. 18–33, Hamburg, Germany, September, 2006.
- [6] Y. Zhang and V. Paxson, “Detecting stepping stones,” *USENIX Security Symposium*, vol. 171, p. 184, 2000.
- [7] A. Blum, D. Song, and S. Venkataraman, “Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds,” in *Proceedings of the Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, RAID*, Sophia Antipolis, France, September, 2004.
- [8] S. J. Murdoch and P. Zieliński, “Sampled Traffic Analysis by Internet-Exchange-Level Adversaries,” in *Proceedings of the Privacy Enhancing Technologies*, pp. 167–183, Ottawa, Canada, June, 2007.
- [9] Y. Sun, A. Edmundson, and L. Vanbever, “RAPTOR: Routing Attacks on Privacy in Tor,” in *Proceedings of the USENIX Security 2015*, Washington, D.C.USA, August, 2015.
- [10] M. Nasr, A. Bahramali, and A. Houmansadr, “DeepCorr: Strong flow correlation attacks on tor using deep learning,” in *Proceedings of the CCS ’18: 2018 ACM SIGSAC Conference on*

- Computer and Communications Security*, Toronto Canada, October, 2018.
- [11] Y. Sun, A. Edmundson, and N. Feamster, "Counter-RAPTOR: safeguarding tor against active routing attacks," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, California, USA, May, 2017.
 - [12] Obfs4, "Obfs4," <https://bridges.torproject.org/bridges?transport=obfs4>.
 - [13] W. D. L. Cadena and A. Mitseva, "Traffic-sliver: fighting website fingerprinting attacks with traffic splitting," in *Proceedings of the . CCS (2020)*, pp. 1971–1985, Virtual Event, USA, October, 2020.
 - [14] G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 186–203, 2017.
 - [15] S. . Henri, G. Garcia-Aviles, and P. Serrano, "Protecting against website fingerprinting with multihoming," *PoPETS*, vol. 2, no. 2020, pp. 89–110, 2020.
 - [16] X. Cai and N. Rishab, "Cs-bufflo: A congestion sensitive-website fingerprinting defense," in *Proceedings of the WPES (2014)*, pp. 121–130, Scottsdale, AZ, USA, November, 2014.
 - [17] X. Cai, "Asystematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the CCS (2014)*, pp. 227–238, Scottsdale, AZ, USA, November, 2014.
 - [18] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "peek-a-boo, I still see you: why efficient traffic analysis countermeasures fail," in *Proceedings of the IEEE S&P (2012)*, pp. 332–346, IEEE, San Francisco, CA, USA, May, 2012.
 - [19] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proceedings of the Computer Security - ESORICS 2016*, pp. 27–46, Springer, Heraklion, Greece, September, 2016.
 - [20] T. Wang and I. G. Walkie-talkie, "An efficient defense against passive website fingerprinting attacks," in *Proceedings of the of USENIX Security*, pp. 1375–1390, San Diego, CA, USA, August, 2017.
 - [21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, March, 2016.
 - [22] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical blackbox attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, April, 2017.
 - [23] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," 2015, <https://arxiv.org/abs/1412.6572>.
 - [24] S. M. M. Dezfouli, A. Fawzi, F. Omar, and F. Pascal, "Universal adversarial perturbations," in *Proceedings of the The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Juan, PR, USA, July 2017.
 - [25] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (sp)*, May, 2017.
 - [26] D. Song, K. Eykholt, I. Evtimov et al., "Physical adversarial examples for object detectors," in *Proceedings of the 12th USENIX Workshop on Offensive Technologies (WOOT 18)*, USENIX Association, Vancouver, BC, Canada, August, 2018.
 - [27] Y. Shi, S. Wang, and Y. Han, "Curls & whey: boosting black-box adversarial attacks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, WA, USA, June, 2019.
 - [28] N. Carlini and D. Wagner, "Audio adversarial examples: targeted attacks on speech-to-text," in *Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW)*, May, 2018.
 - [29] Q. Yao, N. Carlini, G. Cottrell, I. Goodfellow, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *Proceedings of the 36th International Conference on Machine Learning, Volume 97 of Proceedings of Machine Learning Research*, PMLR, California, USA, June, 2019.
 - [30] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, and F. Koushanfar, "Universal adversarial perturbations for speech recognition systems," in *Proceedings of the Interspeech*, Seattle, WA, USA, September 2019.
 - [31] J. Ebrahimi, A. Rao, L. Daniel, and D. Dou, "Hotflip: white-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Melbourne, Australia, July, 2018.
 - [32] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," 2018, <https://arxiv.org/abs/1711.02173>.
 - [33] P. Neekhara, S. Hussain, S. Dubnov, and F. Koushanfar, "Adversarial reprogramming of text classification neural networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, June, 2019.
 - [34] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015, <https://arxiv.org/abs/1412.6572>.
 - [35] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2016, <https://arxiv.org/abs/1607.02533>.
 - [36] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2016, <https://arxiv.org/abs/1608.04644>.
 - [37] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2574–2582, Seattle, WA, USA, June, 2016.
 - [38] T. consensus, "Tor consensus," 2006, <https://gitweb.torproject.org/torspec.git/tree/path-spec.txt>.
 - [39] T. Rendezvous, "Tor rendezvous specification version 3," 2017, <https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt>.
 - [40] DeepCorr Dataset, "DeepCorr Dataset," 2018, <http://traces.cs.umass.edu/index.php/Network/Network>.
 - [41] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: undermining website fingerprinting defenses with deep learning," in *Proceedings of the CCS (2018)*, pp. 1928–1943, Toronto, Canada, October, 2018.
 - [42] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proceedings of the NDSS*, San Diego, California, USA, February, 2018.
 - [43] P. Winter, T. Pulls, and J. F. ScrambleSuit, "A polymorphic network protocol to circumvent censorship," in *Proceedings of the WPES '13 Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, New York, NY, USA, August 2013.
 - [44] S. Xia, B. Chen, and G. Wang, "Two classification methods based on a novel multiclass label noise filtering learning framework," *IEEE Transactions on Neural Networks and Learning Systems*, no. 1–15, 2021.
 - [45] X. Yu, T. Liu, and M. Gong, "Transfer learning with label noise," 2017, <https://arxiv.org/pdf/1707.09724.pdf>.

- [46] S. Xia, S. Zheng, G. Wang, X. Gao, and B. Wang, "Granular ball sampling for noisy label classification or imbalanced classification," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2021.
- [47] T. Liu and D. Tao, "Classification with noisy labels by importance reweighting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 3, pp. 447–461, 2016.
- [48] M. Numan, F. Subhan, W. Z. Khan et al., "A systematic review on clone node detection in static wireless sensor networks," *IEEE Access*, vol. 8, Article ID 65450, 2020.
- [49] Z. Guo, L. Tang, T. Guo, K. Yu, M. Alazab, and A. Shalaginov, "Deep Graph neural network-based spammer detection under the perspective of heterogeneous cyberspace," *Future Generation Computer Systems*, vol. 117, pp. 205–218, 2021.
- [50] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating DNN-based traffic analysis systems in real-time with Blind adversarial perturbations," in *Proceedings of the of USENIX Security*, San Diego, CA, USA, August, 2021.

Research Article

A Lightweight Flow Feature-Based IoT Device Identification Scheme

Ruizhong Du, Jingze Wang , and Shuang Li

School of Cyber Security and Computer Science, Hebei University, Bao Ding, Hebei, China

Correspondence should be addressed to Jingze Wang; wjz9921@gmail.com

Received 5 October 2021; Revised 17 December 2021; Accepted 28 December 2021; Published 15 January 2022

Academic Editor: Weiwei Liu

Copyright © 2022 Ruizhong Du et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Internet of Things (IoT) device identification is a key step in the management of IoT devices. The devices connected to the network must be controlled by the manager. For this purpose, many schemes are proposed to identify IoT devices, especially the schemes working on the gateway. However, almost all researchers do not pay close attention to the cost. Thus, considering the gateway's limited storage and computational resources, a new lightweight IoT device identification scheme is proposed. First, the DFI (deep/dynamic flow inspection) technology is utilized to efficiently extract flow-related statistical features based on in-depth studies. Then, combined with symmetric uncertainty and correlation coefficient, we proposed a novel filter feature selection method based on NSGA-III to select effective features for IoT device identification. We evaluate our proposed method by using a real smart home IoT data set and three different ML algorithms. The experimental results showed that our proposed method is lightweight and the feature selection algorithm is also effective, only using 6 features can achieve 99.5% accuracy with a 3-minute time interval.

1. Introduction

With the popularization and development of high-speed networks, artificial intelligence, big data, and other technologies, the number of IoT (Internet of Things) devices connected to the Internet has also rapidly increased. According to Cisco's forecast, there will be 500 billion IoT devices by 2030 to access the Internet [1]. The mounting number of IoT devices poses threats to the network [2] and brings more challenges to network managers [3]. In Cisco's recent comprehensive report on network security [4], it was stated that an increasing number of hackers utilize the vulnerabilities of IoT devices to carry out cyberattacks. In the current Internet environment, exploiting IoT devices to implement DDoS (distributed denial of service) attacks has become a primary form of attack [5]. Therefore, learning how to manage IoT devices and ensuring the security of the IoT network system have become the issues of most concern for network managers.

Presently, there are methods to ensure the security of IoT systems by authenticating IoT devices through cryptographic approaches or deep learning [6]. However, these

methods are generally costly and unsuitable for the characteristics of low energy consumption and low computing power of networked devices, which will affect the performance of IoT system's effectiveness. At the same time, the traditional anomaly detection system judges whether the device exhibits abnormal behavior by detecting the abnormality of the traffic pattern. However, the Internet of Things devices have massive and heterogeneous characteristics, and it is unmanageable to identify abnormal data behavior patterns. Therefore, identifying the types of IoT devices connected to the network is of great significance to the management of IoT devices, especially in a low cost way. In the case of limited gateway computing resources, efficiently and accurately identifying devices is a problem that needs to be urgently solved.

To better identify devices on the gateway, this study proposed a lightweight IoT device identification method based on flow features. This solution studies the flow-related statistical characteristics intensively; then to pursue less cost, a novel NSGA-III-based [7, 8] filter type feature selection algorithm is proposed; and finally, the extra random tree algorithm is used to build a device recognition model to

classify devices. The features used in this paper are elaborated: first, the features are at the transport layer, so this method is suitable for all IoT devices that communicate on TCP/IP protocol stacks; second, they also do not include plaintext features, effectively avoiding the problem of feature invalidation caused by encrypted transmission and at the same time efficiently perform feature extraction, model construction, and IoT device identification; last, the proposed novel feature selection method also plays an important role in reducing the cost through the device identification process.

Some of the important contributions of our present work are listed below:

- (1) To solve the problem of IoT device identification in a low-overhead manner, we develop a lightweight IoT device identification scheme based on feature selection and machine learning algorithms. We also demonstrate its ability to identify IoT devices with over 99.5% accuracy with less cost than other schemes.
- (2) In-depth research has been carried out on flow-related statistical characteristics and the time interval of feature extraction. DFI technology is used to build features to avoid the unavailability of plaintext features due to data encryption and improve the performance of feature extraction, model construction, and device identification.
- (3) Based on NSGA-III, we introduce symmetric uncertainty and correlation coefficient and propose a novel low-overhead feature selection method to perform feature selection on the extracted flow-related statistical features in IoT device identification, and the valid features are filtered while reducing the dimensionality of the features.
- (4) Experiments are conducted on a real data set. The experimental results show that the proposed feature selection method performs well and the proposed scheme can achieve higher accuracy in a short time window. Its cost is much lower than the existing method. It can also achieve the same accuracy as the actual scheme.

The remainder of this paper is arranged as follows: Section 2 demonstrates the related works. In Section 3, we explain our proposed feature selection method and the IoT device identification model. In Section 4, we exhibit the experimental results and data set. Finally, Section 5 contains the conclusion of this work.

2. Related Works

Recently, researchers have proposed a variety of solutions for identifying IoT devices. The current IoT device identification schemes can be classified into two categories from the perspective of fingerprint acquisition methods: one is the active detection method, and the other is the passive traffic analysis method. The active detection method obtains the response by sending requests to the target device and

extracts the banner for device identification by analyzing the content of the response. The passive method extracts features by analyzing the daily traffic generated by the device. Feng et al. [9] proposed an active detection method for device discovery and identification, which uses the application layer response generated by the device to extract the banner and builds a fingerprint database and then establishes the map between device response and device type, vendor, and model. They achieved a very fine-grained device identification scheme, but this approach needs to send massive packets to the network, which will bring huge cost to the devices. These methods focus more on device discovery rather than management. To better manage IoT devices and offer low cost, our proposed method extract feature is passive.

Miettinen et al. [10] proposed a framework to identify the types of networked devices and restrict the communication of vulnerable ones. They used 23 features generated from the traffic packets of the IoT devices to construct fingerprints for each device. A classifier was trained for each device type to identify vulnerable devices. This method can differentiate vulnerable devices from normal devices easily, but they only detect whether the devices are normal when they are first introduced into the network. This approach is not intended for long-term device management. We resolve this problem in our method by continuously collecting the traffic devices produced. Furthermore, Marchal et al. [11] proposed AuDI, which divides the network traffic into “flows,” which are several time series. They defined the flow as the traffic that uses a specific protocol to communicate with a MAC address. When a packet is sent in 1 second, it is marked as 1 in the time sequence. Then, the DFT (discrete Fourier transform) periodic features of traffic are calculated and obtained, and 33 features related to traffic cycle are used to classify the devices combined with the kNN algorithm. This novel method uses DFT to construct features, but the features have high dimension, and the DFT process also introduces much cost to the identification system. However, our method avoids these expensive calculations.

Santos et al. [12] utilized the four statistical features of traffic characteristics combined with the text information of the user agent extracted from the packet payload and the random forest algorithm to classify the devices. Le et al. [13] proposed a method for device classification based on DNS traffic. They extract the content of DNS traffic packets, using the TF-IDF algorithm for feature construction to classify and identify the type, vendor, and model of the device. Msadek et al. [14] proposed a dynamic sliding window traffic segment method, and they used DPI (deep packet inspection) technology for feature construction and a variety of machine learning algorithms for model construction and evaluation. These three methods use plaintext features for device identification. Nevertheless, for encrypt traffic, these features will be invalid. Our method avoids using plaintext features for this reason. Aksoy and Gunes [15] proposed a method using GA (genetic algorithm) to reduce the dimension of the feature vector and utilized a variety of machine learning algorithms to build a secondary classification model to classify devices in a genre-model granularity. Nonetheless,

the GA algorithm and secondary classification introduce more cost into the system. Shahid et al. [16] used the size of the first n packets and $N - 1$ time intervals of TCP session interaction between devices as features and various machine learning algorithms for device identification. This method is also not suitable for long-term device management. There are also some research developing device identification schemes based on signal process, like [17, 18]; their research focuses on physical layer performance of the devices, which is not our point, but as effective methods in IoT device identification, we also consider their works.

The main contributions of these studies were to construct special features associated with device type accomplishing device type identification by machine learning algorithms. The features are essential in this type of work. Sivanathan et al. [19] deeply investigated the characteristics of traffic in a flow level, and they constructed a 2-stage classifier for device classification. In the first stage, they extract DNS queries, port number, and cipher suits from these text features to obtain a class and confidence value. In the second stage, they combined the output of the first stage and flow-level statistics with random forest to classify devices. We used their method as a baseline method for comparison. Based on their work, we optimized the feature selection to reduce the IoT device identification system's cost, attaining a lightweight method with comparable identification accuracy.

3. The Proposed Device Identification Model

The system model is pictured in Figure 1. First, we take the captured traffic as input and select a fixed time interval to split the traffic; second, we generate flows from the split traffic, extract flow-level features by a statistical method, and then filter out invalid and redundancy features by the proposed feature selection method, which is based on NSGA-III; finally, a variety of machine learning algorithms and the features selected in the previous step are integrated to classify devices and multiple time intervals are selected for experimentation. The most suitable time interval and machine learning algorithm is then selected to build the efficient device classification model.

3.1. Feature Description. The purpose of this article is to build an efficient and accurate IoT device identification scheme based on flow-related statistical features for device identification. The first step for device identification is using flow statistical values to represent the behavior of IoT devices. In addition, the method in this paper selects the flow generated in a fixed time window, which prevents the problem of low efficiency of feature extraction caused by a flow of too long duration. At the same time, it was found that when the bilateral flow is used for feature extraction, the features generated by the large amount of flow data produced from the frequent mutual access of devices in the LAN will decrease classification accuracy. This is mainly because the frequent mutual access of the devices generates a large amount of the same traffic, which results in similar features.

For example, the traffic between the Belkin Wemo switch and motion sensor in the data set has this problem.

Table 1 shows the result of address statistics on the pcap data of the Belkin Wemo motion sensor using Wireshark. DstIpAddress represents the destination IP address of the packets, and Count is the count of packets. 192.168.1.223 is the IP address of the Belkin Wemo switch. 64.14% of the traffic is accessing each other, which will produce a large number of similar features, leading to the deterioration of the device identification model. In view of the fact that a large number of network attacks require access to the Internet, the flow features used in this solution are all bidirectional flows when local devices interact with external network services or devices.

Flow [19] is identified by a five-tuple group: source IP address, destination IP address, source port, destination port, and protocol. The related statistical characteristics of flow are flowVolume's (the sum of bytes of two-way flow upload and download) median, mode, maximum, minimum, information entropy, mean and variance, flowRate's (flowVolume/duration of flowVolume) the same statistics as flowVolume. At the same time, the port number accessed by the device can also be used as a part of the basis for classification. To fit the machine learning algorithm, the port number-related features are processed as follows in this scheme: first, the port numbers are classified into three categories: the port numbers 0–1023 are assigned to certain services as one category, represented as port1; 1024–49151 are loosely bound to the port numbers of some services as a category, represented as port2; 49151–65535 dynamic or private ports are in a category, and binary encoding is performed on this three categories, represented as port3. The number of occurrences of the port number is recorded, denoted as port1Cnt, port2Cnt, and port3Cnt. Moreover, the number of occurrences of flows that belong to different protocols (TCP/UDP) is recorded, denoted as (udpCnt, tcpCnt).

For ease of deployment, this solution extracts flow-related information within a fixed time window as classification features. The choice of time window will affect the effect of the solution. When the time window is short, the overhead of storing and extracting features is small. However, in a short period of time, the flow statistical characteristics of some devices show high similarity, which will lead to a decrease in the accuracy of the model; when a long-time window is selected, the storage and extraction of the features will be costly, but the flow statistical features of different devices relatively deviate from each other. Therefore, it is necessary to make a trade-off between the storage and extraction feature overhead and the classification accuracy. The gateway device is sensitive to the storage and calculation overhead, so the time window should be shortened appropriately.

3.2. Feature Selection. The purpose of feature selection is to select a valid subset of attributes and to remove irrelevant or redundant attributes. Traditional feature selection methods can be divided into three categories, namely the filter, wrapper, or embedded methods. Compared with the other

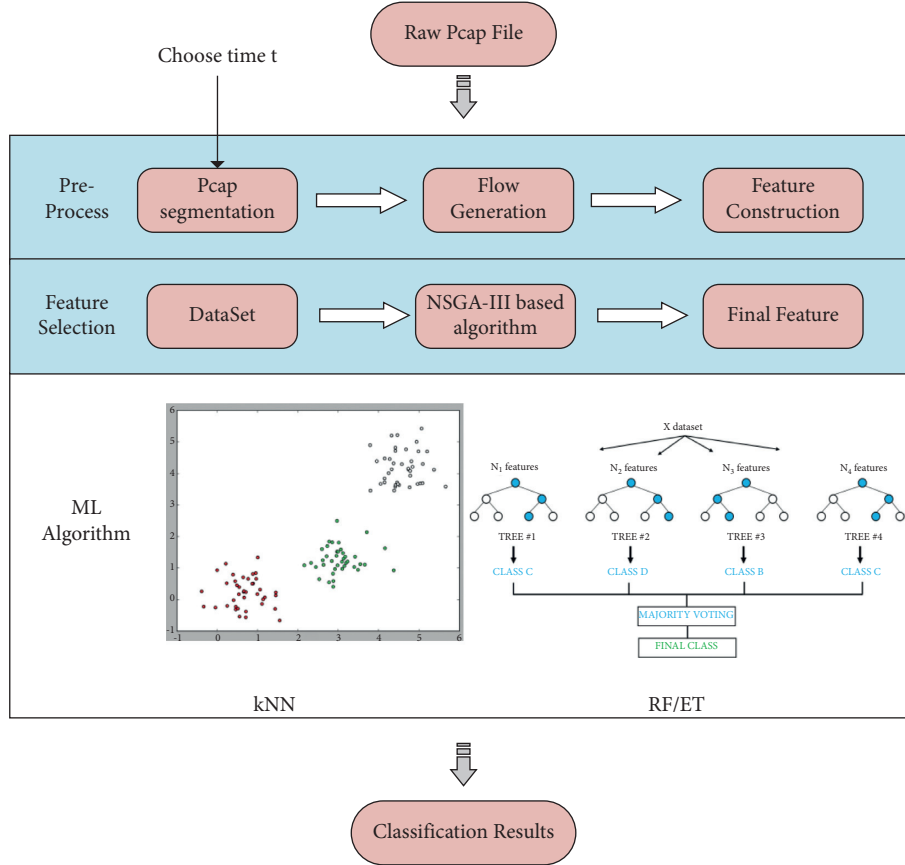


FIGURE 1: System model.

TABLE 1: Belkin Wemo motion sensor traffic statistics.

DstIp Address	Count	Percent (%)
239.255.255.250	1339	1.58
192.168.1.249	15 006	17.72
192.168.1.223	54 303	64.14
192.168.1.208	3403	4.02
192.168.1.1	8124	9.60
184.73.174.14	1494	1.76
174.129.217.97	992	1.17

two types of methods, the filter method does not require machine learning algorithm training in the feature selection process and is the least expensive method of the three. The filter method assumes that the selected optimal feature combination is a set of valid features. How to evaluate the utility of the feature is a key issue in the filter method. To better ensure the effect of selecting features, a feature selection method based on multiple objective functions using NSGA-III is proposed.

To ensure the effectiveness of features, this method models feature selection as a multiobjective optimization problem and uses NSGA-III to search for the optimal solution. There are three objective functions/evaluation functions. In the following description, F represents the set of all the features, SF represents the selected feature subset,

and NSF represents the unselected feature subset, which have the following relationship:

- (1) $F = SF \cup NSF$
- (2) $SF \cap NSF = \emptyset$

3.2.1. Symmetric Uncertainty [20] Based Objective Function. Mutual information (MI) of two variables is a measure of the degree of interdependence between variables. The value of mutual information represents the degree to which the uncertainty of the other variable is reduced when one variable is known. Mutual information $MI(X; Y)$ between two random variables X and Y is shown in equation (1).

$$MI(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_b \frac{p(x, y)}{p(x)p(y)}. \quad (1)$$

The value of b is 2, $p(X)$ and $p(Y)$ are the probability density functions of X and Y , respectively, and $p(X, Y)$ is the joint probability density function of X and Y . Symmetric uncertainty is standardized mutual information, which makes the information shared between random variables comparable, and it is always used in the feature selection process. The calculation of symmetric uncertainty is exhibited by using equation (2).

$$SU(X, Y) = 2.0 \times \frac{MI(X; Y)}{-\left(\sum_x p(x) \log p(x) + \sum_y p(y) \log p(y)\right)}. \quad (2)$$

The value range of $SU(X, Y)$ is between 0 and 1. The closer the symmetric uncertainty value is to 1, the more relevant the variables X and Y are. At this point, we obtain the first objective function, which is represented by using equation (3).

$$F_1 = \frac{\sum_{f_i, f_j \in SF, f_i \neq f_j} SU(f_i, f_j)}{\sum_{f \in SF, c = \text{class}} SU(f, c)}, \quad (3)$$

$SU(f_i, f_j)$ is the symmetric uncertainty between feature i and feature j in SF, and $SU(f, c)$ is the symmetric uncertainty between feature f and class in SF. The smaller the function value, the better the classification effect of feature set SF.

3.2.2. Correlation Coefficient-Based Objective Function.

Correlation coefficient is also a method used to measure the degree of correlation between variables. The difference between symmetric uncertainty and the correlation coefficient is that the latter measures the degree of correlation between variables from the perspective of statistics, while the former measures the degree of correlation from the perspective of information entropy. The calculation of the correlation coefficient is shown in equation (4).

$$\text{COR}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}, \quad (4)$$

$\text{cov}(X, Y)$ is the covariance of random variables X and Y , and σ_X and σ_Y are the standard deviations of X and Y , respectively. We can design the second objective function, defined as equation (5).

$$F_2 = \frac{\sum_{f_i, f_j \in SF, f_i \neq f_j} \text{COR}(f_i, f_j)}{\sum_{f \in SF, c = \text{class}} \text{COR}(f, c)}, \quad (5)$$

f_i, f_j, c have the same meaning as equation (3). The smaller the function value, the better the classification result of feature set SF. To enable the feature selection method to achieve the purpose of dimensionality reduction, the third objective function is introduced by using equation (6).

$$F_3 = |\text{SF}|. \quad (6)$$

3.2.3. NSGA-III Algorithm. The framework of the NSGA-III [7, 8] algorithm is roughly the same as the NSGA-II algorithm. The main difference lies in the individual selection mechanism of the offspring: NSGA-II selects the offspring based on the crowding distance, and NSGA-III uses the method based on reference points. NSGA-III solves insufficient algorithm convergence and diversity when multi-objective optimization problems with three or more

objective functions are involved. The algorithm also makes it easier to find the optimal solution.

To optimize the proposed three objective functions (F_1, F_2, F_3), the steps of the NSGA-III algorithm are as follows:

- (1) Generate an initial population that has N individuals. Individuals are a sequence of random values between 0 and 1. A value larger than 0.5 represents a selected feature, otherwise, the feature is not selected.
- (2) Generate reference points set Z^* based on the three objective functions.
- (3) Use evolutionary operators to generate a child population and evaluate objective values for every individual.
- (4) Use nondominated sort for the combination of father and child populations.
- (5) Nondominated ranking based on Pareto dominance on the combined population.
- (6) Select N individuals as the next generation based on the former reference point set.
- (7) Repeat (3)–(6) until it reaches the maximum iteration times to obtain the Pareto optimal solution set.

3.3. Machine Learning Algorithm. To achieve the best results, we selected three machine learning algorithms based on their descriptions in literature [21], evaluating them from the perspectives of accuracy and training speed and selecting the best performing algorithm to ensure that the method proposed in this article has a higher classification accuracy with less overhead.

The following briefly introduces the three machine learning algorithms used in the experiment:

- (1) *k-Nearest Neighbor (kNN) Algorithm.* kNN is a classification algorithm with no training process. The most important parameter is k , if the input sample x is given, x will be classified into the k samples closest to x in the training set for most samples in the same category. kNN is used in the preliminary experimental verification process.
- (2) *Random Forest (RF).* RF is an ensemble learning method that contains multiple CART decision trees. There have been many articles using RF to construct the IoT device identification scheme that achieved excellent results, indicating it is suitable for the device identification system.
- (3) *Extremely Randomized Trees (ET).* ET is very similar to RF. The difference between this method and RF is that the selection of the node bifurcation attributes of the decision tree in ET is random, while the node division in RF of the bifurcation attribute is selected after Gini index calculation. Given its high similarity with RF, we select this algorithm as a part of the device identification system for comparison with the RF's results.

4. Data Set, Experiment Results, and Analysis

In this section, we will conduct a detailed analysis of the used data set [19] and the selected features of this scheme and use different machine learning algorithms at different time intervals to evaluate the classification results and cost. Finally, the best performing ML algorithm is given, and the model is constructed based on this algorithm.

The experimental environment is a personal computer, the detailed configuration is Intel core i5 9400 2.90 GHz, memory 8 GB, win10 64-bit operating system. The experimental steps are as follows: first, the collected data are subpackaged at fixed time intervals, and then the joy tool [19] is used to extract the flow information; second, Python script is used to calculate the relevant statistical values from the output of joy and constructs the features for storage and finally uses the machine learning algorithm provided by scikit-learn [22] to establish machine learning models and classify the devices and evaluate the classification results.

4.1. Data Set. The data set used in this article comes from the public data set of the paper [19], which is obtained by collecting the traffic of smart home devices in the laboratory under the campus network environment. The IoT devices in the data set include cameras, smart lighting tools, activity sensors, and health monitors. The TP-Link router acts as a gateway through which all devices connect to the Internet. In the data collecting progress, they connect to the router through an additional device, use tools such as tcpdump to passively collect the traffic of all devices, and save the traffic collected every day as a pcap file, which is stored in the hard disk connected to the device. This article uses opened 20-day data for experiments. Because the solution in this paper is based on the characteristics of the transport layer construction and classification, the provided data set only gives the mac address corresponding to the device, and we also analyze the IP address corresponding to the devices.

4.2. Feature Selection Results. This solution uses the filter feature selection method based on NSGA-III to remove redundant features while reducing the dimensionality of the features, that is, to reduce the computational cost of the model while ensuring the accuracy of the classification. NSGA-III is a variant of the GA algorithm. For individual construction: the number of elements contained in the individual is the same as the cardinal of the full set of features; initially, the value of each element is a random number between 0 and 1, and an element greater than 0.5 represents the feature is selected. When conducting the experiment, the number of individuals used is 40, and the number of iterations is set to 100. We performed feature selection on the 1-min time interval for small overhead introduced to the system. Figure 2 shows the results of NSGA-III operation.

As can be seen in Figure 2, the results appear to have the minimum value of three at the same time. The features selected in the Pareto front are port2 (destination port between 1024 and 49151), port2Cnt, tcpCnt, udpCnt, the mode of flowVolume, and the variance of flowVolume. The

time complexity of NSGA-III is $O(N^2M)$, where N is the number of individuals in the population (40) and M is the number of objective functions (3). The feature selection process only brings less additional overhead to the system. Through our feature selection method, we select six features from the 22 features we described in Section 3.1. For our objective to be lightweight, this approach markedly reduces the classification and training overhead.

We also compared the features used in this research and the baseline method, and the features and the selection status are shown in Table 2. Our purpose is to deeply investigate the applicability of flow-related statistics and establish a lightweight IoT device identification scheme; therefore, we construct the feature set almost from the flow-related statistics because it is easy to get the flow-related statistics, which means the feature extraction progress only bring little cost to the system. The baseline method just uses the mode of flow volume and flow rate and then also forms word bag models for port, domain name, and cypher suit, and these text features are imported to a Bayes classification to generate the class and probability for final classification. From a lightweight point of view, we only use one-level classifier and remove the text features on account of text features need to be processed additionally and cause extra cost. In the selection process, the features are also selected properly to further cut the cost. At the same time, the classification performance can be maintained above a high level, and the classification details are shown in the following.

About the selected features after feature selection progress, we attempt to explain why our feature selection algorithm chooses them. First, port2 and port2Cnt represent the devices access the port between 1024 and 49151, users' customized services always run on these ports, as different devices access different services, the access times and whether access these ports should show great discrimination between devices. The variance and mode of flow volume represent the quantity of device traffic and the fluctuation of traffic, and they describe device communication behavior from the traffic view. And for the TCP and UDP flow counts, they represent the protocol discrimination between the devices, as different devices access different services, the flows always use different protocols, and these features describe the devices' behavior from the view of protocol. Combine all selected features, we can describe the device communication behavior comparatively comprehensively, and therefore, the classification results can reach a high level on accuracy.

4.3. Classification Results. In this section, we will evaluate our scheme mainly from two points of view. The first is the classification performance, which is used to measure the applicability of an IoT identification method, and to prove our scheme's lightweight characteristics, the second view is the cost of our method.

4.3.1. Classification Performance. The following will show the results of classifying the data set using the three machine learning algorithms mentioned before and the features

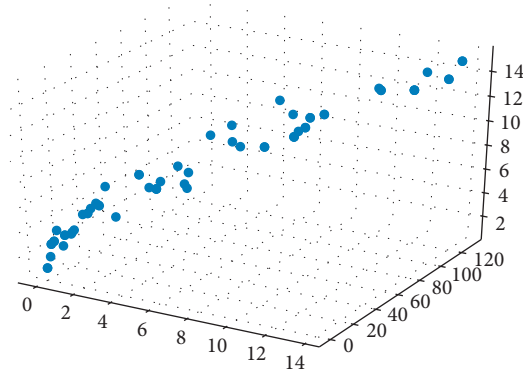


FIGURE 2: Pareto optimal fronts obtained by NSGA-III.

TABLE 2: Features used in this research.

Name	Explanation	Selection status		
		Before selection	After selection	Baseline selection
VOL_MED	Flow volume's median	✓		
VOL_MOD	Flow volume's mode	✓	✓	
VOL_MAX	Flow volume's maximum	✓		
VOL_MIN	Flow volume's minimum	✓		
VOL_IE	Flow volume's information entropy	✓		
VOL_AVG	Flow volume's average	✓		
VOL_VAR	Flow volume's variance	✓	✓	✓
RATE_MED	Flow rate's median	✓		
RATE_MOD	Flow rate's mode	✓		
RATE_MAX	Flow rate's maximum	✓		
RATE_MIN	Flow rate's minimum	✓		
RATE_IE	Flow rate's information entropy	✓		
RATE_AVG	Flow rate's average	✓		
RATE_VAR	Flow rate's variance	✓		✓
PORT1	Whether the flow access port between 0 and 1023 appeared	✓		
PORT2	Whether the flow access port between 1024 and 49591 appeared	✓	✓	
PORT3	Whether the flow access port between 49592 and 65535 appeared	✓		
PORT1_CNT	The count of remote IP port between 0 and 1023	✓	✓	
PORT2_CNT	The count of remote IP port between 1024 and 49591	✓		
PORT3_CNT	The count of remote IP port between 49592 and 65535	✓		
UDP_CNT	The count of flows use UDP	✓	✓	
TCP_CNT	The count of flows use TCP	✓	✓	
DUR_MOD	Flow duration's mode			✓
SLP_TIME	Time intervals' mode between flows			✓
DNS_INT	DNS intervals' mode			✓
BAG_PORT_NUM	Word bag model of port which flow accessed			✓
BAG_DOMAIN	Word bag model of DNS domain names			✓
BAG_CS	Word bag model of cipher suit			✓

obtained by feature selection progress. 80% of the data are used as the training set, and the remaining 20% as the test set. We conducted experiments and evaluations at intervals of 1 min, 2 min, 3 min, 4 min, 10 min, 30 min, and 1 hour. For every algorithm, we use 10-fold cross-validation to ensure the result is stable and repeatable. Evaluation indicators include model training time and classification results related to the evaluations. The following indicators were used when evaluating the classification results:

- (1) Precision: $Pr = TP/TP + FP$
- (2) Recall: $Re = TP/TP + FN$
- (3) Accuracy: $Acc = TP + TN/TP + FP + TN + FN$
- (4) F_1 score: $F_1 = 2 \times Pr \times Re/Pr + Re$

TP represents the number of positive examples correctly classified in the data, FP is the number of positive examples incorrectly classified, TN is the number of negative examples correctly classified, and FN is the number of negative examples incorrectly classified.

Due to the selected algorithms having hyperparameters, different parameters will have an impact on the accuracy and training speed of the model. RandomizedSearchCV [22] is used in the parameter selection to ensure that the performance of the model in each time interval is the best. The accuracy shown in Figure 3 is the result obtained on the test set. It can be seen that, for the performance of accuracy, the longer the time interval, the greater the deviation of characteristics in the streams of different devices, which brings better classification results. When the time interval is longer than 3 min, the accuracy of the RF and baseline method is stable at about 99.5%. However, a decrease occurred for the kNN algorithm. As we inspected the feature set used in the training, we found that as the time segment became longer, the feature extract frequency became lower, so the feature set became smaller. For the kNN algorithm, the result is strongly dependent on the scale of the feature set unlike the other algorithms. However, in a comparable time segment, the performance of kNN is much worse than that of the other algorithms. To prove that our scheme is statistically better than the baseline method, we conduct 100 times of training and prediction on a 1-minute time segment. As shown in Figure 4, the accuracy of our scheme is statistically 1.5% higher than that of the baseline method.

As shown in Figures 5 and 3, in a short time window, our method's classification performance is better than the baseline method's. As we inspect the features, the DNS interval, NTP interval, and sleep time that the baseline method used are meaningless in a short time interval, but the features chosen in our method always are meaningful. In other words, with a short time segment, some features in the baseline method especially time interval features become homogenized and are inadequate to discriminate different devices. But the features used in our method, constructed from flow-related statistics and selected after the NSGA-III-based feature selection method, are adequate to distinguish devices whether the time segment is long or not.

We also present the detailed classification performance on 3-min time segment because as shown in Figure 5 the

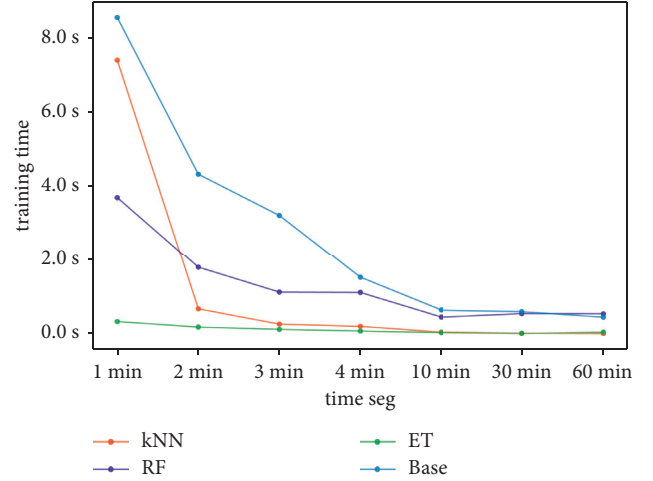


FIGURE 3: Time cost of different algorithms.

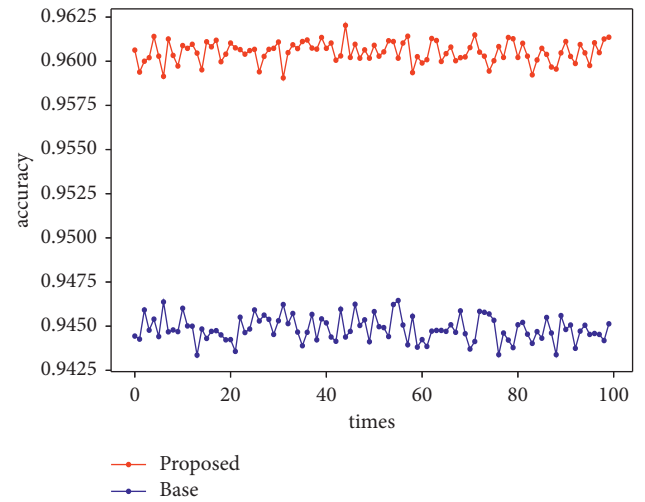


FIGURE 4: 100 times test.

accuracy will increase and reach a peak value till the time segment is 3 min. As Table 3 shows, our proposed method based on RF and ET can reach a comparative level with the baseline method. The results show our method's strength clearly: comparative or superior classification performance and much less overheads, which will be clarified in the following.

4.3.2. Overhead of Proposed Method. In terms of training time, the training time of ET is always the shortest, as the time intervals become longer, the shorter the time cost to train the model, and this is mainly for longer time intervals, making the feature set smaller. We should notice that when evaluating the training time of the baseline method, only the time for the second-level classification is considered. The first-level classification will generate a label and a degree of confidence for each sample, and this process will cause heavy cost especially for an enormous data set.

Our method also uses less storage space after feature extraction; as shown in Table 4, as the time intervals become

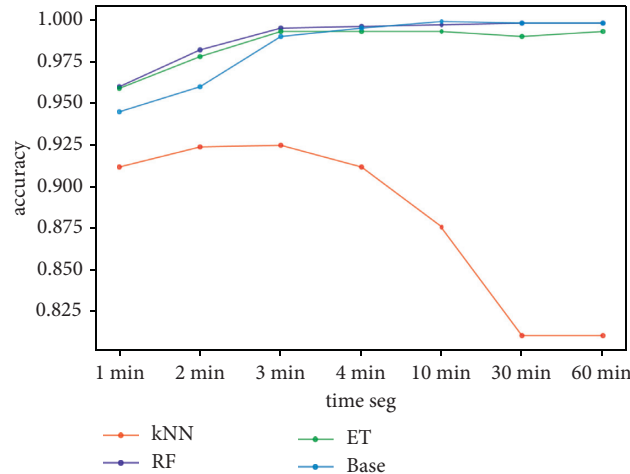


FIGURE 5: Accuracy of different algorithms.

TABLE 3: Detailed evaluation on 3 min.

Algorithm	Accuracy	Precision	Recall	F_1 score
kNN	0.912	0.912	0.913	0.912
Base	0.994	0.993	0.994	0.994
RF	0.995	0.995	0.995	0.995
ET	0.993	0.993	0.993	0.993

TABLE 4: Storage usage.

Time Seg	Proposed (MB)	Baseline (MB)
1 min	35	75
2 min	22.2	62.3
3 min	17	49
4 min	12.1	35.3
10 min	7.4	20
30 min	4.1	16
1 hour	3.3	13.4

longer, the storage used by the proposed method is much less than the baseline method, and this is mainly caused by the text features used in the baseline method. Therefore, our method is superior to the baseline method on storage cost.

Whether in terms of training time or feature dimension, our scheme achieves better performance with less cost. We also obtained a detailed evaluation when the time interval was 3 min. As shown in Table 2, the performance of ET using the selected features in this article was very close to that of the baseline method, while the overhead was significantly reduced. The accuracy of ET is close to the best, which RF achieved, but ET's training is much faster than RF, and on the basis of trade-off on time cost and classification accuracy, we proved that ET is also a valid algorithm to construct an IoT device identification scheme.

5. Conclusion

As the popularization of IoT devices are connected to the Internet, managing and annotating these devices is an essential problem for keeping network security. In this paper, we propose

a lightweight IoT device identification scheme based on traffic analysis. This scheme used flow-related statistical features to represent the behavior of IoT devices and a filter feature selection method based on NSGA-III to select effective features. Machine learning algorithms are used to classify devices. Experimental results showed that our proposed scheme can achieve comparable accuracy with much less overhead. Based on the ET algorithm combined with the six attributes port2, port2Cnt, tcpCnt, udpCnt, flowVolume's mode, and flowVolume's variance, the best classification result can be achieved, and the training speed is the fastest. When the time interval is 1 min, an accuracy of 95.8% can be achieved, while the accuracy of the base method is only 94.5%. As for a long time interval like 3 min, our method can achieve an accuracy of 99.3%. At the same time, the overhead is greatly reduced compared with the base method. This method is suitable for deployment on the gateway to identify IoT devices. Future work will focus on cloud services. How to integrate the models, ensure the trustworthiness of the gateway, and improve the performance and security of the distributed device identification system will be the focus of future work.

Data Availability

The data set is the same as the paper “Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics” used and the access link is <https://iotanalytics.unsw.edu.au/>.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61972073), the Key Program of Natural Science Foundation of Hebei Province of China (F2019201290), and the Natural Science Foundation of Hebei Province of China (F2018201153), Post-Graduate’s Innovation Fund Project of Hebei University.

References

- [1] Cisco, “Internet of things: connected means informed,” Technical Report, 2020, <https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.html>.
- [2] A. Riahi Sfar, E. Natalizio, Y. Challal, and Z. Chtourou, “A roadmap for security challenges in the internet of things,” *Digital Communications and Networks*, vol. 4, no. 2, pp. 118–137, 2018.
- [3] M. A. Khan and K. Salah, “IoT security: review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.
- [4] Cisco, “Annual cybersecurity report,” Technical Report, 2018, <https://www.cisco.com/c/dam/m/digital/elq-cmcglobal/witb/acr2018/acr2018final.pdf>.
- [5] H. Griffioen and C. Doerr, “Examining mirai’s battle over the internet of things,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 743–756, New York, NY, USA, October 2020.
- [6] Y. Huang, W. Wang, H. Wang, T. Jiang, and Q. Zhang, “Authenticating on-body iot devices: an adversarial learning approach,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5234–5245, 2020.
- [7] K. Deb and H. Jain, “An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: solving problems with box constraints,” *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2013.
- [8] H. Jain and K. Deb, “An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part ii: handling constraints and extending to an adaptive approach,” *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 602–622, 2013.
- [9] X. Feng, Q. Li, H. Wang, and L. Sun, “Acquisitional rule-based engine for discovering internet-of-things devices,” in *Proceedings of the 27th USENIX Security Symposium*, pp. 327–341, Baltimore, MD, USA, August 2018.
- [10] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, “IoT sentinel: automated device-type identification for security enforcement in iot,” in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2177–2184, IEEE, Atlanta, GA, USA, June 2017.
- [11] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, “AuDI: toward autonomous IoT device-type identification using periodic communication,” *IEEE Journal On Selected Areas In Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.
- [12] M. R. Santos, R. M. Andrade, D. G. Gomes, and A. C. Callado, “An efficient approach for device identification and traffic classification in iot ecosystems,” in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00304–00309, IEEE, Natal, Brazil, June 2018.
- [13] F. Le, J. Ortiz, D. Verma, and D. Kandlur, “Policy-based identification of IoT devices’ vendor and type by DNS traffic analysis,” in *Proceedings of the Policy-Based Autonomic Data Governance*, pp. 180–201, Springer, Berlin, Germany, 2019.
- [14] N. Msadek, R. Soua, and T. Engel, “IoT device fingerprinting: machine learning based encrypted traffic analysis,” in *Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–8, IEEE, Marrakesh, Morocco, April 2019.
- [15] A. Aksoy and M. H. Gunes, “Automated IOT device identification using network traffic,” in *Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, Shanghai, China, May 2019.
- [16] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, “IoT devices recognition through network traffic analysis,” in *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data)*, pp. 5187–5192, IEEE, Seattle, WA, USA, December 2018.
- [17] Y. Liu, J. Wang, J. Li et al., “Zero-bias deep learning for accurate identification of internet-of-things (iot) devices,” *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2627–2634, 2021.
- [18] B. Charyyev and M. H. Gunes, “Locality-sensitive iot network traffic fingerprinting for device identification,” *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1272–1281, 2021.
- [19] A. Sivanathan, H. H. Gharakheili, F. Loi et al., “Classifying iot devices in smart environments using network traffic characteristics,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.
- [20] S.-Y. Jiang and L.-X. Wang, “Efficient feature selection based on correlation measure between continuous and discrete features,” *Information Processing Letters*, vol. 116, no. 2, pp. 203–215, 2016, <https://www.sciencedirect.com/science/article/pii/S0020019015001271>.
- [21] M. Woźniak, M. Grana, and E. Corchado, “A survey of multiple classifier systems as hybrid systems,” *Information Fusion*, vol. 16, pp. 3–17, 2014.
- [22] F. Pedregosa, G. Varoquaux, A. Gramfort et al., “Scikit-learn: machine learning in python,” *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Research Article

IoT-IE: An Information-Entropy-Based Approach to Traffic Anomaly Detection in Internet of Things

Yizhen Sun,^{1,2} Jianjiang Yu,³ Jianwei Tian,^{1,2} Zhongwei Chen,^{1,2} Weiping Wang,³
and Shigeng Zhang^{3,4}

¹State Grid Information & Communication Company of Hunan Electric Power Corporation, Changsha, China

²Hunan Key Laboratory for Internet of Things in Electricity, Changsha 410004, China

³School Computer Science and Engineering, Central South University, Changsha 410012, China

⁴State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Correspondence should be addressed to Shigeng Zhang; sgzhang@csu.edu.cn

Received 19 October 2021; Accepted 14 December 2021; Published 30 December 2021

Academic Editor: Jinwei Wang

Copyright © 2021 Yizhen Sun et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Security issues related to the Internet of Things (IoTs) have attracted much attention in many fields in recent years. One important problem in IoT security is to recognize the type of IoT devices, according to which different strategies can be designed to enhance the security of IoT applications. However, existing IoT device recognition approaches rarely consider traffic attacks, which might change the pattern of traffic and consequently decrease the recognition accuracy of different IoT devices. In this work, we first validate by experiments that traffic attacks indeed decrease the recognition accuracy of existing IoT device recognition approaches; then, we propose an approach called IoT-IE that combines information entropy of different traffic features to detect traffic anomaly. We then enhance the robustness of IoT device recognition by detecting and ignoring the abnormal traffic detected by our approach. Experimental evaluations show that IoT-IE can effectively detect abnormal behaviors of IoT devices in the traffic under eight different types of attacks, achieving a high accuracy value of 0.977 and a low false positive rate of 0.011. It also achieves an accuracy of 0.969 in a multiclassification experiment with 7 different types of attacks.

1. Introduction

With the popularity of the Internet of Things (IoTs) [1], the number of devices connected to the Internet has grown rapidly. According to the report released by Ericsson in June 2020 [2], it is expected that the number of cellular IoT connections will exceed 5 billion by 2025, nearly three times of that in 2020. IoT applications have penetrated into many fields such as medical, agriculture, and logistics, providing efficient collaborative work for human production, life, and home travel [3, 4]. The IoT security has aroused extensive attention in academia and industry. From the industrial IoT, to the vehicle IoT, and then to the smart-home IoT, there are a large number of security requirements in any scenarios. However, due to the use of weak keys, the security problems caused by security flaws in the design, as well as the user's

weak security awareness, etc., IoT devices are vulnerable to malicious intrusion by criminals [5], leading to the disclosure of user's private data, or even the breakdown of industrial control systems, resulting in a large amount of direct and indirect economic losses.

IoT devices' identification based on traffic is an important safety measure to maintain and control device assets. It has two advantages [6]: one is introducing machine learning technology into device identification task to achieve automated identification; the other is the difficulty of obtaining traffic data is much easier than other data. You only need to install the captured API on the router to capture the two-way communication traffic between the IoT devices and the remote servers in real time. In particular, compared with controlling devices to actively send request packets to obtain the desired information, it is almost zero cost to

monitor its communication in a passive manner, and no prior knowledge is required. Existing IoT device identification methods can be roughly divided into encrypted traffic identification and unencrypted traffic identification. However, according to Gartner's report, more than 80% of the enterprise network traffic has been encrypted by 2019, and this is an irreversible trend. Thus, in recent years, many researches have begun to focus on the IoT device identification based on encrypted traffic. Since the contents of the payload in application layer cannot be extracted because of the encrypted traffic, some statistical features would inevitably be utilized under normal circumstances, including (1) packet size; (2) packet arrival time interval (IAT) [7]; and (3) frequency domain features of periodic time series traffic. These selected features and machine learning algorithm based on statistical features have achieved good results.

However, we find that the existing researches [8–10] are focused on how to improve the identification accuracy, without considering the situation of IoT devices being attacked by malware. In fact, the two tasks of IoT device identification and anomaly detection are complementary. The malicious traffic is mixed with the benign traffic, and if it is not detected and filtered, it will inevitably affect the development of the identification work. This also leads to the lack of robustness of most current identification methods. As long as a few malicious traffic is added, it will greatly reduce the identification accuracy. Our experiment found that only 10% of the malicious traffic was added to the 9 types of IoT device traffic collected in the laboratory, and the identification accuracy dropped from 99% to 75%. Thus, how to design an anomaly detection model of IoT devices based on encrypted traffic is of critical significance.

Our goal is not to detect malware [11], but to detect whether IoT devices are under attack. Existing solutions for abnormal traffic detection of IoT devices fall into two types according to the granularity of division. The detection granularity of the first solutions is accurate to each packet [12, 13]. Its disadvantage is that the identification accuracy is often very low and needs to extract a large number of attributes for each packet to form a high-dimensional feature vector. The second type of solutions detects whether the device has been attacked within a period of time [14–16]. It usually uses statistical features over a period of time to achieve anomaly detection. Although the real-time performance is not as good as the first type of solutions, the selection of features is more reasonable and the accuracy rate is higher.

To improve the detection accuracy as much as possible on the basis of ensuring real-time performance, we propose an anomaly detection model IoT-IE based on information entropy and sliding window. Obviously, it is an improvement on the second type of solutions. Information entropy is a statistic that describes the value distribution of a variable. Here its most critical feature is that the changes in the number of values that rarely occur have a greater impact on the entropy than the changes in the number of values that frequently occur; that is, values that rarely occur play a significant role in IoT-IE, which matches the phenomenon that the attribute value of abnormal packet is different from

the same attribute value of normal packet exactly. Hence, we can effectively filter the window of abnormal traffic before the task of IoT devices' type identification and evaluate the retention rate of abnormal packets as well as the loss rate of normal packets through the anomaly detection model.

Contribution Summary. Our aim is to design an IoT devices' anomaly detection model based on encrypted traffic, IoT-IE. The main contributions are as follows:

- (1) Aiming at the real 7 types of attacks on IoT devices, verify the impact of the traffic attack model for IoT devices on the accuracy of device identification.
- (2) Comparing the normal and abnormal behavior patterns of traffic from vulnerable devices, we proposed a traffic monitoring method based on information entropy and sliding window for specific device types.

On the public IoT device malicious traffic dataset, our anomaly detection method can distinguish benign traffic and abnormal traffic with an accuracy of 97.73%, which is better than the current baseline method, and still achieves a good result in identifying the attack types.

2. Related Works

In this section, we first discuss the IoT device identification for unencrypted traffic and encrypted traffic based on whether the IoT device traffic is encrypted or not. After that, we will introduce some real-world attacks and threats against IoT devices, as well as the existing anomaly detection researches for IoT device traffic

2.1. IoT Device Identification

2.1.1. Unencrypted Traffic. The main characteristic of IoT plaintext traffic is that it can extract high-level payload, which makes a huge contribution for device type identification.

The paper [17] proposes a fingerprinting generation method for discovering IoT devices in the cyberspace. It selects TCP/IP/UDP header field values and words extracted from application layer data as features to generate fingerprinting of IoT devices. Feng et al. design an Acquisitional Rule-based Engine (ARE) [18]; it extracts the key fields in the application layer in response packets as the search query of the crawler website and utilizes the Named-Entity Recognition (NER) to extract the device labels from the selected web pages, and finally the association algorithm is utilized to generate the annotation rules for the IoT devices and to discover IoT devices in the cyberspace through these rules.

2.1.2. Encrypted Traffic. Although the payload of the traffic is an intuitive and efficient feature for device type identification, it will cause an infringement of user privacy and a higher cost in feature extraction. In addition, identification methods based on payload characteristics have become

infeasible in the research of IoT devices identification for encrypted traffic.

Radhakrishnan et al. [19] observe the heterogeneity between devices. They first propose utilizing IAT to identify IoT devices by capturing device traffic and extracting the statistical characteristics of IAT to create the unique signature of the device type; then through Artificial Neural Network (ANN), they identify the extracted device fingerprints to realize accurate classification of the device type. Aneja et al. [20] extract this feature to draw IAT diagrams for every 100 packets as well and utilized Convolutional Neural Networks (CNN) to process the generated IAT diagrams.

Msadek et al. [21] utilize dynamic segmentation technology on encrypted flow, extract relevant statistical distribution such as protocol type, packet size, and number of packets as features and then compare and evaluate five types of machine learning algorithms: KNN, Support Vector Machine, Random Forest, AdaBoost, and Extra-Tree. The work of Pinheiro et al. [22] is similar, but they only utilize three characteristics of the mean and standard deviation of packet length generated by IoT devices in one second, and the total number of bytes sent in this second. However, it is only for the known device identification, the method seems powerless for the new IoT devices that is increasing in number at this stage.

2.2. IoT Device Attack and Anomaly Detection. The huge heterogeneity and scale of IoT devices brings severe challenges to device assets management and security protection [23, 24]. At the end of September 2016, the website KrebsOnSecurity.com was hit by a large-scale DDoS attack launched by Mirai. Mirai malware scans services such as Telnet on the network to spread and then uploads its own binary files on the device through the load service to realize infection. The infected devices continue to scan for other vulnerable devices. Finally, the intruder sends control instructions through the C&C server to attack the target [25, 26].

In order to deal with the increasingly severe IoT security issues, especially large-scale DDoS attacks, some related work has been carried out in recent years. In [27], Nguyen et al. implemented an autonomous and self-learning distributed IoT device detection system. They built a device-specific normal behavior model and through the GRU neural network model to detect the deviation of benign flow and malicious flow then isolated Infected devices. For solving the detection of unknown suspicious activities or zero-day attacks, the paper [16] designs a two-stage anomaly detection method based on machine learning. In the first stage, a supervised ML algorithm is used to identify known malicious behaviors. In the second stage, an unsupervised ML algorithm such as clustering is used to identify unknown malicious behaviors or zero-day attacks. This has achieved good results in detecting a wide range of IoT attacks.

Anthi et al. design a supervised IoT device anomaly detection model [12]. They extract the header field value of OSI each layer for each data packet, including a total of 121 features such as packet length, flag, port, and window size, test and evaluate the detection performance of Naive Bayes,

J48, Logistic Regression, Random Forest, SVM, and Fully Connected Neural Network for scanning attacks, DoS attacks, MIMT attacks, replay attacks, and spoofing attacks against IoT devices. However, the disadvantage of this method is that it utilizes a large number of redundant features. On this basis, KS test and Pearson's correlation coefficient are utilized for dimensionality reduction in [13]; only 28 features are utilized to achieve high-precision detection of these attacks finally.

Yair et al. propose a network-based IoT anomaly detection method N-BaIoT [15], which extracts the statistical characteristics of packet length, IAT, and packet number of IoT benign flow in five time windows (the latest 100 ms, 500 ms, 1.5 s, 10 s, and 1 min), and train deep autoencoders (one for each device) to characterize the benign behavior of IoT devices. If the autoencoder is trained on benign samples only, it will successfully reconstruct the normal observations, and when a major reconstruction error is detected, it classifies the given observation as abnormal and finally has achieved good results in terms of accuracy and time. Wan et al. propose an anomaly detection method based on the minimum description length (MDL) principle [28]. They extract features such as flow duration, source and destination IP address, source and destination ports, protocol type, number of packets, number of bytes, and compressing and encoding and then take the encoded length as the abnormal score of the traffic to be measured. Finally, the score threshold is set to detect malicious traffic.

3. Preliminary Work

For a general IoT device-type identification system, if an intruder attacks various devices connected to the network before the IoT device traffic is entered into the identification system, it will inevitably lead to the destruction of the inherent pattern of IoT device traffic, consequently affecting the identification system ability to determine the type of the device under attack. The purpose of this section is to explore the impact of the type and volume of traffic attacks on identification performance.

3.1. Attack Models. IoT devices are mainly subject to two types of attacks. One is a port scanning attack, whose purpose is to discover the open ports of the device to achieve intrusion and then control the IoT devices; the other is a denial-of-service attack, which is mainly to cut off the communication between the device and the remote server and invalidate the functions provided.

Since our IoT device-type identification system and anomaly detection model is based on encrypted traffic, we refer to the threat model mentioned in [14] and consider the following 6 different types of attacks. These attacks do not require the intruder to have a wealth of prior knowledge, only a basic understanding of the attack command.

In order to enrich the abnormal traffic dataset, we consider different reflection/DDoS attacks and ensure the normal operation of the devices during the device identification tasks, we reduce the rate of attack traffic, aiming to

change the traffic pattern of the devices and make the devices classified incorrectly. See Section 4.2.1 for details on how to attack and the process of collecting benign and abnormal traffic.

- (1) *ArpSpoof*. ARP is a protocol that converts IP addresses into physical addresses. ARP spoofing refers to the fact that the intruder sends a forged ARP reply message to the device so that it utilizes wrong information to update the ARP cache table, resulting in communication errors [29].
- (2) *Ping of Death*. Ping of Death is a common denial of service attack. When the IP packet length exceeds the maximum size of the Ethernet frame, the packet will be fragmented and sent as multiple frames. The receiving end can only reassemble after receiving all the fragments. Normally, the reassembled IP packet will not exceed the specified maximum size. Ping of Death is to send a large number of IP packets that exceeding the maximum size, and the extra data in the packet will be written into other normal areas, which will cause buffer overflow.
- (3) *TCP SYN*. TCP SYN flooding mainly occurs at the fourth layer of OSI. The principle of the attack is that after the intruder forges the connection request from the sender and the receiver returns the response packet of the first handshake, it cannot receive the feedback of the third handshake from the sender and finally consumes the server's memory and causes to crash.
- (4) *SNMP Reflection*. SNMP is mainly used to manage the devices in the local area network and can obtain the basic configuration information and status information of the devices. The intruder forges the SNMP request packets of the source IP address, and the reflection servers send a large number of response packets to the victim devices after receiving them, which will cause network congestion [30].
- (5) *SSDP Reflection*. SSDP is commonly applied to Universal Plug and Play (UPnP) devices for device discovery. In view of the limitation of SSDP is that it does not check whether the querier is in the same network as the device, similar to SNMP, it is also a UDP protocol that can easily be used for reflection attacks.
- (6) *Smurf*. The ICMP Echo request packet is utilized to diagnose the network. The device will respond with an ICMP Echo Reply to the source address of the message after receiving it. Once the source address is set to the broadcast address, all devices on the local network must process these broadcast messages. A large number of Reply broadcast messages will cause the network to flood [31].

3.2. Traffic Attacks Decrease Device Identification Accuracy. After reading a large number of literatures on fingerprint identification schemes that are vulnerable to traffic attacks, it is found that packet size and IAT are the two most common

features they used. We adopt the strategy of making things simple, utilizing the above two features and several general machine learning algorithms to identify, and consequently explore the impact of the traffic attack model for IoT devices on the performance of the conventional device identification method.

For a mixed traffic dataset containing benign traffic and malicious traffic, when extracting statistical features for classification tasks, malicious attacks will change the inherent pattern of traffic sent by IoT devices; the statistical features and time series features of traffic will definitely occur changes. After that, whether the traffic is divided according to the session as the sample, or divided by a time window cutting method as the sample, will lead to a decline in the identification rate.

In one-way ARP spoofing, the victim IoT device receives many response packets; the time interval between these response packets remains stable and differs significantly from the normal packets time interval, while the packet size is controlled by the attack script, which can be random or of a specified size but always differs significantly from the normal case. We refer to several traditional machine learning methods mentioned in [6, 21, 22, 32, 33]. Here, we attack devices using the *ArpSpoof*. Figure 1 shows that Random Forest has the best performance in identification accuracy on benign test dataset and identification robustness under attack, which is consistent with the conclusions of [22, 34]. Therefore, the subsequent verification experiments choose Random Forest as the machine learning identification model baseline.

Then, we consider all the attack types for IoT devices mentioned in the previous subsection and take the proportion of the malicious traffic to the total traffic as an independent variable to explore its impact on the identification accuracy of IoT devices.

As shown in Figure 2, it is obvious that as the proportion of the malicious traffic increases, that is, the longer the attack takes, the more obvious the decline in the identification accuracy of IoT devices. At the same time, we find that the top 10% of malicious traffic volume will lead to a sharp decline in the identification accuracy, and then the degree of decline is diminished. In other words, the intruder only needs to inject a small amount of malicious traffic before the device traffic is sent to the identification model to greatly reduce the classification accuracy of the model.

In addition, the type of attack seems to have nothing to do with the decline in the classification accuracy. The reason is that no matter what kind of attack, their traffic patterns are different from the normal traffic. This deviation is the main reason for the decline in classification accuracy.

3.3. Some Observations on IoT Traffic Patterns. Compared with traditional connected devices (PCs and mobile phones), IoT devices are simple in structure. At the beginning of their design, they usually only make use of running a single task or perform a single function. Thus, their traffic always shows a repeated communication mode and generates the same volume of data periodically. Figure 3 shows the distribution

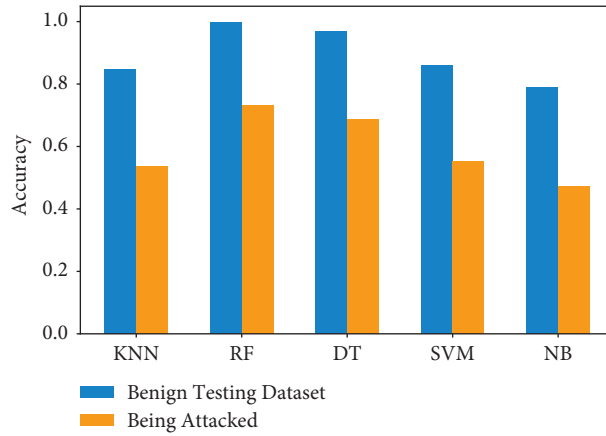


FIGURE 1: The impact of traffic attacks on the identification accuracy.

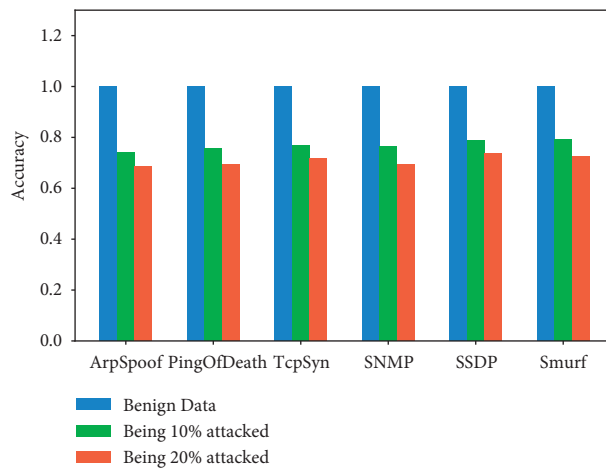


FIGURE 2: The impact of attack type and volume on identification accuracy.

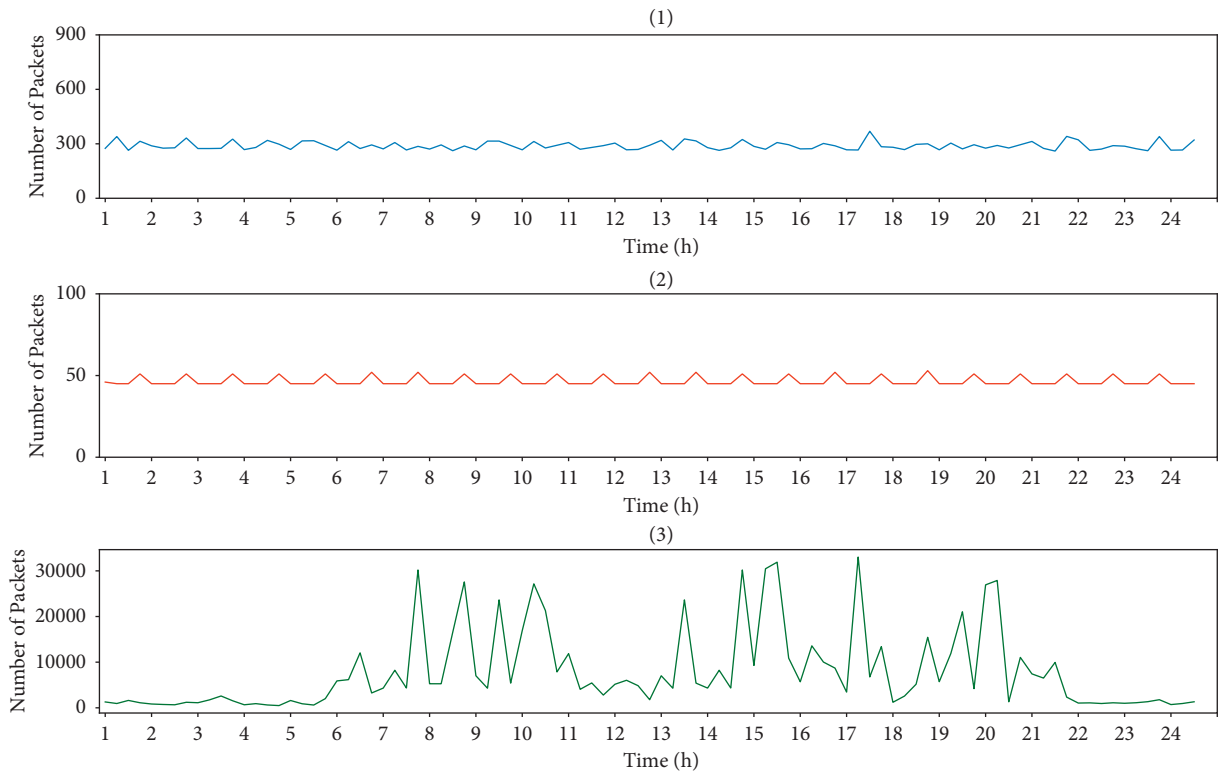


FIGURE 3: IoT/non-IoT device communication volume for one day. (1) Amazon Echo. (2) iHome Smart Plug. (3) Laptop.

of communication volume for two IoT devices and one non-IoT device for one day. Among them, the IoT devices are Amazon Echo and iHome Smart Plug, and the non-IoT device is laptop. It is obvious from the figure that the communication volume of non-IoT device is more irregular than that of IoT devices. This is because all the traffic of laptop comes from the usage records of users, and its communication volume conforms to people's routines. The communication volume in the daytime is much higher than at night.

The packets sending rate of these two IoT devices are in a stable state throughout the whole process for one day, and the number of packets sent within the same size time window tends to be close; here, we assume that IoT devices are in an idle state. And for different types of IoT devices, their packets' sending rates are different. The number of packets sent by Amazon Echo per hour is between 270 and 330, with a fluctuation of about 10%, while the number of packets sent per hour by iHome is less than 60 basically, and it sends 45 packets or 51 packets in most windows, which is related to the specific functions they execute. The functions of smart speakers are far richer than those of smart plugs, and the number of protocols used by smart speakers is also greater than that of smart plugs. So, the average packets sending rate of IoT devices can be used as an important feature for device type identification.

It is known that different destination ports of IoT devices traffic correspond to different protocols/services. Thus, in order to further mine the communication patterns of IoT devices, we continue to find the potential regulations of IoT devices traffic after classifying traffic according to destination port/service type. Figure 4 shows the corresponding data flow after Amazon Echo classified by the most frequently appearing destination ports; the destination ports shown in the figure account for 83% of the total traffic for one day. It can be seen from the figure that the packets' sending frequency belonging to the same service has obvious periodicity: 12 http packets from the device are sent out every 300 seconds, https packets are sent out every 30 seconds basically, and the packets to destination port 33434 is sent every 27 seconds, which is the same as the packets to destination port 49317. The phenomenon is caused by the unique characteristics of IoT devices; they will send packets to their respective servers periodically to remain connected. And it is predictable that the contents of these packets payload are the same basically. From the results, this phenomenon provides a basic follow for the slight statistical differences in each feature between each window after the flow is divided by the sliding window, and the constant traffic pattern can be used as the precondition of anomaly detection.

In addition to describing the IoT devices traffic from the time perspective, we also extract important and commonly used header field values from each packet as spatial information to describe the characteristics of IoT devices traffic. So as to know that the number of attribute values is limited, we draw the Sankey graph to observe. Figure 5 is a Sankey

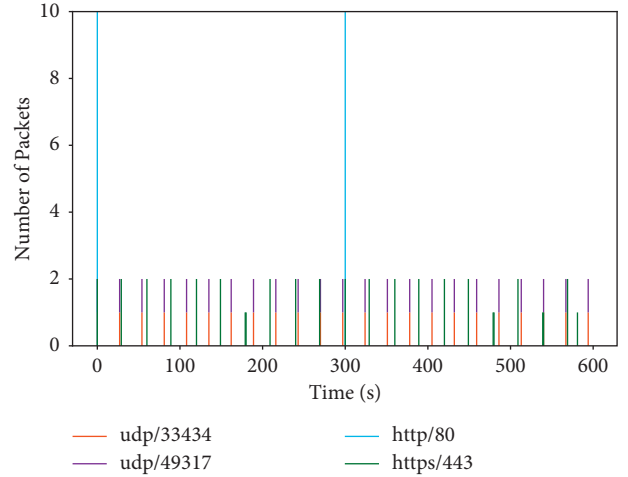


FIGURE 4: Periodic traffic volume of Amazon Echo over a period of time.

diagram of the Amazon Echo traffic (24,609 packets collected over a 24-hour period). The figure shows statistical information, such as the protocol used by the device, the IP address of the remote server it communicates with, and the destination port. We see that Amazon Echo not only involves the necessary 80/443 ports, but also communicates with diverse protocols, a large number of local/external server IP addresses, and different ports that provide various services. It is worth noting that the remote IP associated with port 80 is only 93.184.216.34, while port 443 has several remote IP addresses. Besides, in addition to the common high-level protocol based on UDP, Amazon Echo makes connections to ports 33434 and 49317 of the remote server for maintenance of device-specific services and points to the same remote IP.

4. IoT-IE Overview

In this section, to start with, we introduce the overview and architecture of IoT-IE. Subsequently, we discussed each module of IoT-IE, which work together to monitor IoT device communication.

4.1. System Structure. In the experimental environment we deployed, the gateway acts as a bridge between IoT devices and remote servers, and local IoT devices connect to network via WiFi or Ethernet. Thus, in order to collect IoT devices traffic more conveniently, as well as can be processed and detected in a timely manner at the same time, the IoT gateway also needs to be responsible for hosting our anomaly detection system IoT-IE.

The system architecture of IoT-IE is mainly composed of four key modules: traffic capture, IoT communication data characterization, malicious traffic detection based on information entropy, and alarm/isolation. The traffic capture module is responsible for running the traffic capture commands of IoT devices on the IoT gateway and collects the

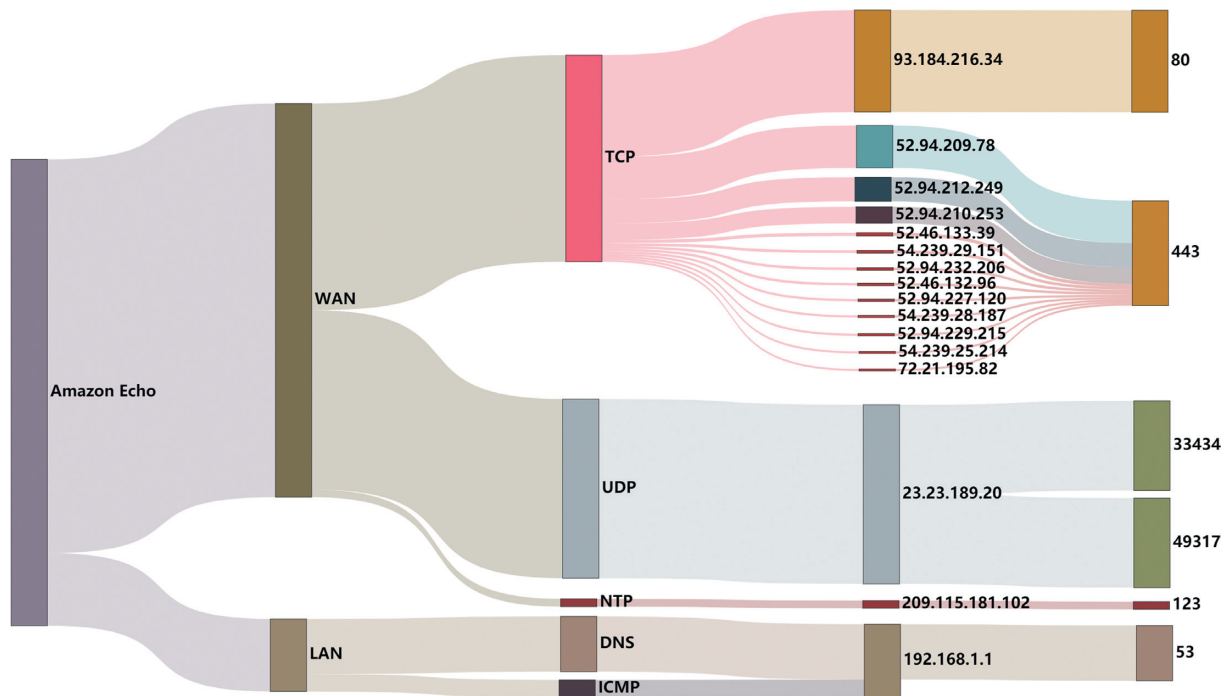


FIGURE 5: Sankey diagram of Amazon echo traffic. Bars from left to right represent device name, WAN or LAN, protocol, server IP address, and destination port. The protocol ICMP has no port.

traffic of the bidirectional communication between the IoT devices and the external cloud servers of IoT providers or other remote servers on the Internet. The main work of the IoT communication data characterization module is to mine the inherent traffic communication patterns of IoT devices in a normal state (without malicious attacks) and build a baseline of IoT devices' benign traffic behaviors, providing basic guidance for subsequent malicious traffic detection work. The task of the malicious traffic detection module based on information entropy is to utilize information entropy as a quantitative metric to measure the statistical differences between benign traffic and malicious traffic so that benign traffic and malicious traffic can be well distinguished, and then machine learning algorithms are used for normal/abnormal binary classification. Finally, the alarm/isolation module isolates the IoT devices that are confirmed to have been attacked so that they cannot communicate with other IoT devices in the experimental environment and prohibit communication with remote servers and alarm at the same time.

4.2. System Models

4.2.1. Traffic capture. As is known to all, the main advantage of an IoT Gateway centric security monitoring system consists in its flexibility to collect all IoT device traffic in a centralized location [16]. Our IoT device traffic collection setup in the laboratory is shown in Figure 6. Use hostapd command on a laptop with Ubuntu Linux operating system to create an IoT gateway, which serves as the access point for the WiFi or Ethernet interfaces of all IoT devices, and then use the traffic capture commands or tools such as TCPdump and Wireshark to capture the traffic data of the bidirectional

communication between IoT devices and IoT provider's cloud servers via the IoT gateway. After that, we refer to the attack scripts published in [14] and use a computer under the local area network to run these scripts to attack the target IoT devices.

4.2.2. Some Observations on IoT Traffic Patterns. The IoT devices that we adopted in the experiment are all consumer IoT devices, which simulate the smart homes' IoT environment. In this module, we mine traffic patterns to prove the difference between IoT devices and traditional connected devices. The predictability of IoT device traffic patterns can make it possible to utilize machine learning to realize type identification and anomaly detection. In general, IoT devices will not be attacked immediately after it is connected to the network, so we consider analyzing the normal communication patterns of IoT devices during the period before they are attacked. This benign traffic can be used for building training set, which can be used for IoT device identification and as negative samples for anomaly detection. Nowadays, personal user privacy is getting more and more attention; the traffic of IoT devices produced by many vendors on the market is encrypted. Thus, we consider the metafeatures of IoT device traffic without any information extracted from the payload of packets, so it does not need rich prior knowledge and feature extraction cost.

Specifically, the traffic features used by IoT-IE can be packet size, IAT, number of bytes, source port, destination port, source IP address, destination IP address, protocol, flow duration, flow average rate, etc. Here, we do not utilize all features but try to explore which features may cause obvious changes after malicious attack to filter out the best

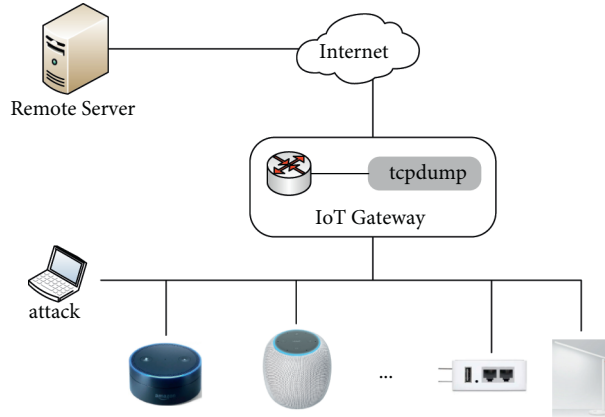


FIGURE 6: IoT device traffic collection platform.

features that can effectively distinguish between benign and malicious traffic. In addition, selecting appropriate features can also reduce the burden of the anomaly detection module and improve the detection efficiency.

4.2.3. Anomaly Detection Based on Information Entropy.

In this module, we propose a malicious traffic detection method based on information entropy and sliding window, as shown in Figure 7. Information entropy is used as a metrics to describe the value distribution of features in a period of time. Different from human-centered Internet traffic, in general, IoT devices repeat the same operations throughout the entire capture process and generate the same amount of data regularly, and each IoT device has its own unique normal communication pattern, so the value of information entropy only fluctuates in a small range over time. However, once malicious traffic appears, some of its attributes will change in the value distribution. For instance, some values that have never appeared before will appear, which is very sensitive to the entropy; it is because the change of information entropy is more sensitive to values with a small probability of appearance than values with a large probability of appearance, resulting in a large difference between the entropy of benign traffic and that of malicious traffic. Then, the inherent feature measurement is sent to the machine learning model for training, and by judging the statistical difference between benign and malicious traffic, whether an attack occurs can be detected.

Alarm/Isolation. Once IoT-IE determines which IoT devices in the experimental environment are under attack, the module will immediately cut off the communication between the infected devices and all other devices in the experimental environment as well as remote servers to take isolation measures and notify people which devices have been attacked by means of alerts.

5. Anomaly Detection Method

Our purpose is to find suitable features or indices to detect whether there is malicious traffic in the IoT network. From the perspective of features value distribution, we can find

differences between the header field values distribution of benign packets and that of malicious packets. Information entropy is a metric that describes the occurrence probability of attribute various possible values, so it can describe the values distribution of each attribute in packets well.

Entropy is a concept in thermodynamics originally, and it is utilized to measure the uncertainty of an attribute in information theory [35].

Firstly, the concept of information quantity is introduced as a measure of “how much” information. The amount of information of a specific event should decrease with its occurrence probability and cannot be negative, so it can be represented by a logarithm, as shown in formula (1). Information entropy is actually the expectation of the amount of information that may be generated before the result comes out. Considering all possible values of the random variable, the expectation of the amount of information that can be brought by all possible events.

$$h(x) = -\log p(x), \quad (1)$$

$$H(X) = \sum_{i=1}^n p(x_i) \cdot h(x_i) = -\sum_{i=1}^n p(x_i) \log(p(x_i)). \quad (2)$$

$p(x)$ represents the probability that the attribute X takes the value x , and there are a total of n possible values for the attribute X .

It can be seen from the previous section that IoT devices have a fixed traffic pattern. If the value of a certain attribute is different from the previous pattern or completely deviates from the pattern, it will cause a huge change in the entropy value. Furthermore, a significant advantage that distinguishes entropy from other statistical features is that it can calculate string-type values, such as IP address. As long as the value distribution is stable, entropy can be effectively used. The entropy feature extraction process is shown in Algorithm 1. For a given window size M and step size T , S samples are generated by cutting flow through a sliding window. For each sample S_i , we extract the appropriate attributes and add them to the corresponding attribute lists, use these lists as input to calculate the entropy value, and finally form the entropy feature vector.

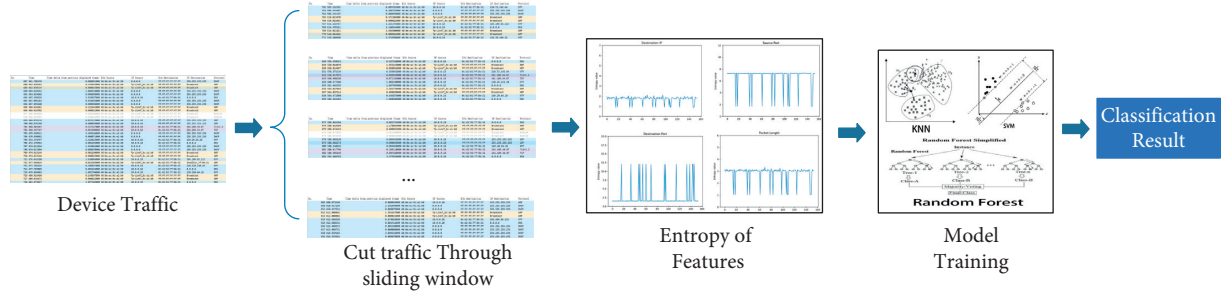


FIGURE 7: Anomaly detection process based on entropy and sliding window.

Input: Raw pcap file, sliding window size M set in advance, sliding step T set in advance, number of attributes to be extracted N .

Output: Feature vector

- (1) $Feature_vector \leftarrow \emptyset$
- (2) Take the first M package according to the window size, move backward T steps each time to form $F_1, F_2, F_3, \dots, F_s$, s represents the number of samples.
- (3) **for** each F_i **do**
- (4) $attribute1_list, \dots, attributeN_list \leftarrow [, K, \dots]$
- (5) **for** each packet P in F_i **do**
- (6) $attribute_1, \dots, attribute_N \leftarrow$ Extract attributes, such as packet size, source port, destination port, destination IP, etc. in P
- (7) **for** each attribute A in $attribute_i$ **do**
- (8) $attribute1_list, \dots, attributeN_list \leftarrow \cup A$
- (9) **end for**
- (10) **end for**
- (11) **for** each $attribute_list$ in $attributeI_list$ **do**
- (12) $Entropy \leftarrow F2(attribute1_list)$, which is mentioned in Section 5
- (13) $Feature_vector \leftarrow Feature_vector \cup Entropy$
- (14) **end for**
- (15) **end for**
- (16) **return** $Feature_vector$

ALGORITHM 1: Entropy feature extraction.

We take part of the IoT devices in the experiment to calculate the entropy value of common features, including the normal communication traffic of IoT devices and the abnormal communication traffic represented by Smurf attacks; the benign communication traffic of IoT devices comes from Amazon Echo. The initial value of the sliding window is set to 300 seconds. As shown in Figure 8, the information entropy in the sliding window for Smurf attack is quite different from that in the same size window for benign traffic. For instance, the entropy value of the packet length basically fluctuates between 3 and 4, and the waveforms of each attribute are relatively similar, indicating that each attribute value of the IoT device corresponds to each other. The entropy values of the destination IP address, source port, destination port, and packet length under Smurf attack are much smaller than the entropy values of the corresponding feature in benign traffic. We delineate a dashed line as a threshold to isolate the benign traffic window and the malicious traffic window. Thus, we can realize malicious traffic monitoring by calculating the deviation degree between the entropy values of the malicious packets attribute field and the corresponding attribute entropy values of the benign packets.

6. Experiment

6.1. Datasets Description. We experimented on the public dataset UNSW-2018 [14]. The UNSW-2018 dataset contains benign traffic and malicious traffic of IoT devices; these two types of data are unbalanced, while the benign traffic is too much. In order to eliminate the impact of unbalanced dataset during training and testing, we take a part of benign dataset only. The authors designed two attack modes: direct attacks (e.g., ARP spoofing, TCP SYN flooding, Fraggle (UDP flooding), and Ping of Death) and reflection attacks (e.g., SNMP, SSDP, TCP SYN, and Smurf), which involve some protocols such as ARP, TCP, UDP, ICMP, DNS, and so on. In order to ensure that the devices remain functional during attack and reflect the attack traffic to the infected devices, a total of 200 attacks were launched at different rates, each attack lasting 10 minutes.

6.2. Evaluation Metrics. We consider the accuracy, precision, TPR (True Positive Rate), FPR (False Positive Rate), FNR (False Negative Rate), and $F1$ -score as the evaluation metrics and adaptability measurement of the test result.

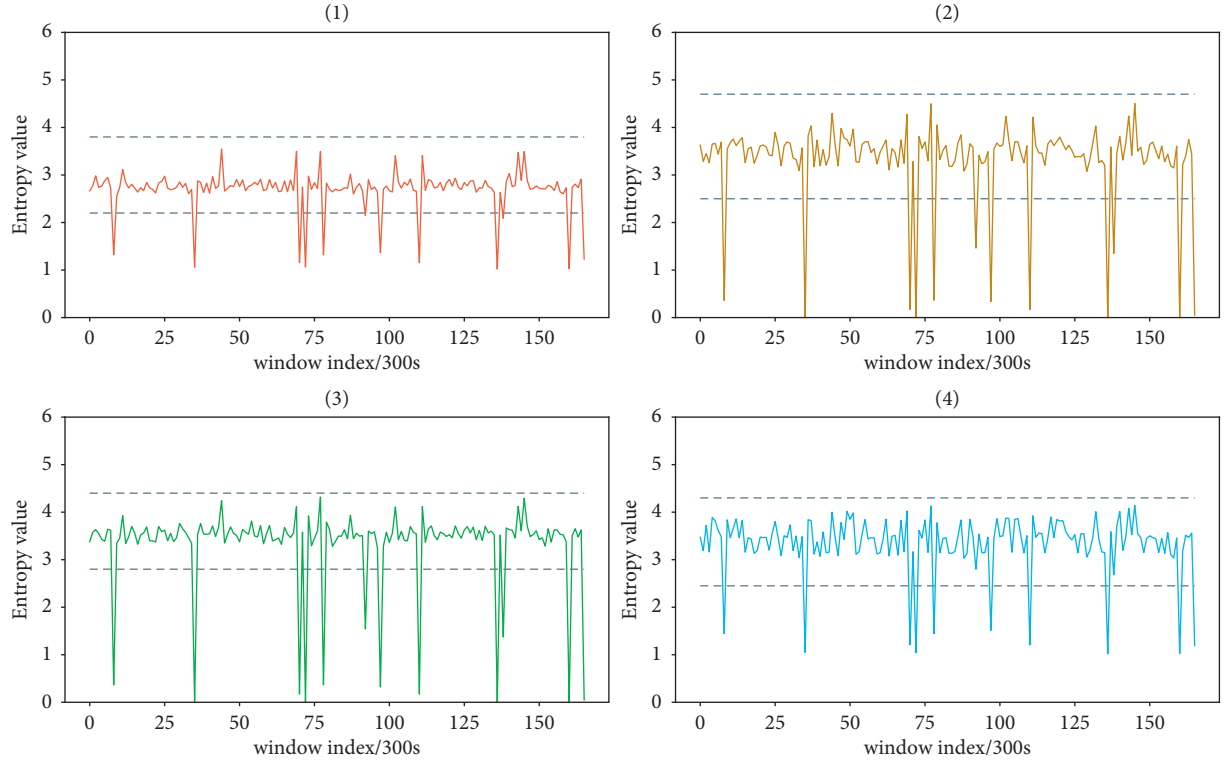


FIGURE 8: Comparison of the benign traffic window entropy and the Smurf window entropy from Amazon Echo in (1) destination IP, (2) source port, (3) destination port, and (4) packet length.

$$\begin{aligned}
 \text{Acc} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \\
 \text{TPR} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\
 \text{FPR} &= \frac{\text{FP}}{\text{FP} + \text{TN}}.
 \end{aligned} \tag{3}$$

Among them, TP represents the number of abnormal flow samples that are correctly classified as abnormal type of flow (True Positives), FN is the number of abnormal flow samples that are incorrectly classified as benign type of flow (False Negatives), and TN represents the number of benign flow samples that are correctly classified as benign type of flow (True Negatives); FP represents the number of benign flow samples that are incorrectly classified as abnormal type of flow (False Positives).

FPR measures the rate that incorrectly classified a sample of benign flow as abnormal flow, which can raise false alarms. TPR represents the percentage of abnormal flow samples correctly classified to abnormal type. Thus, the designed abnormal traffic detection system needs to maximize TPR when FPR is as low as possible, so as to prevent users from being overwhelmed by a large number of false alarms and failing to effectively perform the alarm function of the detection system. On this basis, it is also necessary to implement accurate identification of abnormal traffic in order to satisfy the basic requirements of a good abnormal traffic detection system.

7. Results

We choose IoTArgos [16] as the comparison work of our proposed anomaly detection methods because the difference between them is that the extracted features are completely different, even if they are statistical features over a period of time, IoTArgos mostly utilizes average IAT, average packet size and flow volume, etc., while we utilize entropy features, the subsequent detection algorithms are the same.

As shown in Figure 9, we set the sliding window to 200 in advance, comparing the detection metrics of the two. Compared with IoTArgos, our proposed method IoT-IE has a 1% to 2% improvement in accuracy, precision, and TPR, although the IoTArgos method has more than 95% data on various metrics already. Besides, the result of IoT-IE on FPR is much better than IoTArgos, from 0.025 to 0.007, which is essential for the normal operation of a detection system.

Secondly, we evaluated the performance of our detection method when considering various IoT devices individually, and assume that the window size is 200. The result is illustrated in Table 1 from part of devices. Different algorithms show different detection performance on various types of devices. On the whole, Naive Bayes algorithm is overall inferior to the other four algorithms, and the performance gap of the other four algorithms is very small. This shows that the entropy feature can represent the differentiator between benign traffic and malicious traffic, which has good adaptability to most machine learning algorithms.

For LiFX, all algorithms tested can approach almost 100% detection accuracy, as did for the TP-Link Plug. This is

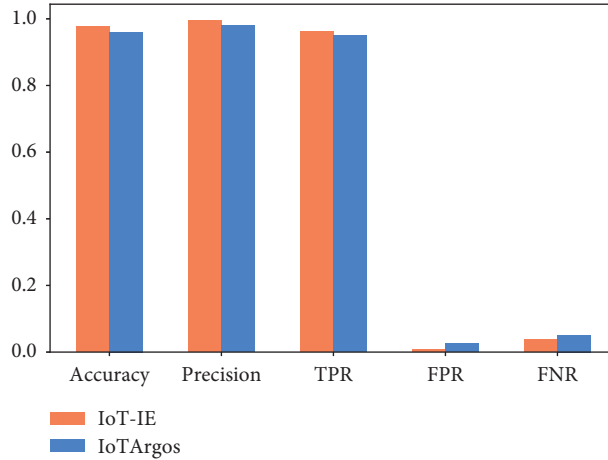


FIGURE 9: Performance comparison of IoT-IE and IoTArgos under 200 size window.

TABLE 1: Detection performance for individual devices.

Algorithm	AE			TP			NC			CU			LX		
	Acc (%)	TPR (%)	FPR (%)	Acc (%)	TPR (%)	FPR (%)	Acc (%)	TPR (%)	FPR (%)	Acc (%)	TPR (%)	FPR (%)	Acc (%)	TPR (%)	FPR (%)
KNN	98.5	97.0	0.0	98.9	98.7	0.4	96.8	95.3	1.6	94.6	91.0	1.7	99.6	99.2	0.0
DT	98.5	100.0	3.1	98.2	99.1	4.5	95.2	94.6	4.2	93.3	91.7	5.0	99.4	98.9	0.0
RF	99.5	99.0	0.0	99.4	99.4	0.4	96.5	95.1	2.1	94.2	92.0	3.6	99.4	100.0	1.2
SVM	100.0	100.0	0.0	99.1	99.0	0.4	97.5	96.5	1.6	94.3	91.1	2.5	99.4	99.2	0.4
NB	99.5	99.0	0.0	98.9	99.0	1.2	85.9	88.8	17.0	86.1	89.3	53.2	99.2	98.9	0.4

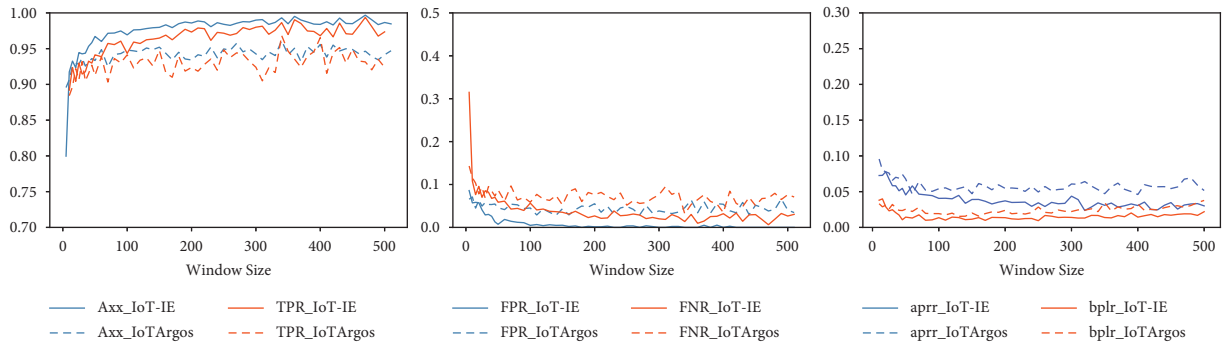


FIGURE 10: Detection performance varying with window size.

related to the function they have. The bulb only has two operations of light on and light off, while the plug only has power off and power on. In contrast, IoT devices such as camera or media player have more functions. As a result, the former has a much smaller change in the benign traffic pattern than the latter.

Thirdly, we explore the influence of sliding window size on detection performance. With the increase of window size, the evaluation metrics of detection performance tend to be stable quickly. When the window size exceeds 40, the accuracy can also rise to more than 95%. Moreover, the method we proposed is always better than IoTArgos obviously after curve tends to be stable.

The two metrics we first proposed are abnormal packets retention rate and benign packets loss rate. The former

means the ratio of abnormal packets in windows which are misclassified as benign traffic windows, and the latter refers to the ratio of benign packets in benign traffic windows which are misclassified to be malicious traffic windows. As shown in Figure 10, its curve with the window size is closely related to the detection accuracy; when the detection is not accurate, there will be more abnormal packets retained and normal packets discarded due to misclassification. In addition, we find that the curve rises slightly when the window size is larger than 400; more packets are retained and discarded due to misclassification in a window, while the total number of packets remains unchanged, resulting in a larger proportion.

Finally, we utilize entropy features to classify the attack types, the classification result is 96.9%, and confusion matrix

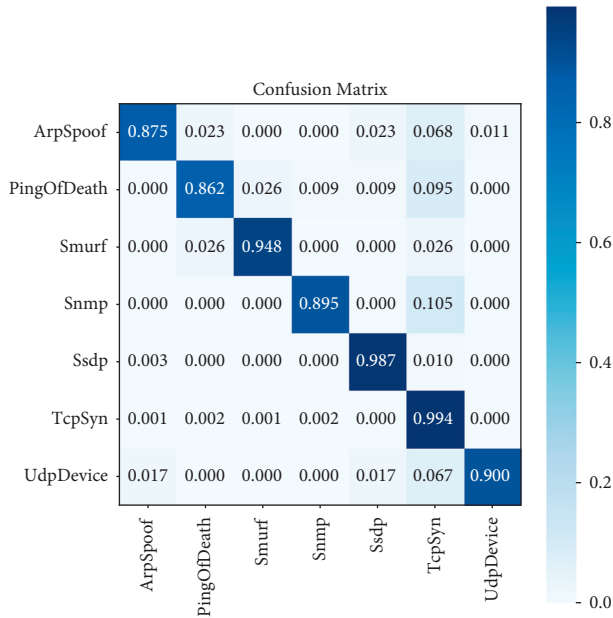


FIGURE 11: Confusion matrix of attack type classification (x-axis: predicted labels, y-axis: true labels).

is shown in Figure 11. Among them, the results of ArpSpooF and Ping of Death are not satisfactory. Through the analysis of the original malicious traffic dataset and its annotation documents, ArpSpooF will affect all features, and zero is added to the missing value in the preprocessing. While it is learned from the annotation document that TCP SYN attack in the dataset affects a specific port of a specific IP, their entropy on IP and port will be very close, which is the same for the Ping of Death.

8. Conclusion

This article introduces a method for detecting anomaly traffic of IoT devices based on information entropy. Firstly, we start from the traffic characteristics of IoT devices and compare with non-IoT devices to highlight the unity and distinguishability of IoT devices in communication patterns. Then, we propose to utilize information entropy and sliding window to detect and locate the malicious traffic of IoT devices, utilizing information entropy to describe the statistical differences of packet attributes and seeking the best classification performance by constantly changing the size of window. Experiments show that our method can still reach an accuracy of 97.73% in response to various types of IoT attacks and has good real-time performance. Even if the window size is compressed to about 40, the detection accuracy can also reach 95%.

Since our method is deployed in a smart home IoT environment currently, the focus of future work is to deploy IoT-IE in power IoT scenarios to evaluate its detection efficiency in different scenarios, mining the inherent communication patterns of IoT device traffic in different scenarios, achieving differentiation in the feature selection, and improving the robustness of anomaly detection.

Data Availability

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

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants nos. 61772559, 61902434, and 62172154, and the Natural Science Foundation of Hunan Province, China, under Grant no. 2019JJ50826.

References

- [1] X. Liu, J. Yin, S. Zhang, B. Xiao, and B. Ou, "Time-efficient target tags information collection in large-scale RFID systems," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2891–2905, 2021.
- [2] V. Hemalatha, K. A. Kumar, M. S. D. P. Gomathi, and Dr. P. Gomathi, "CAMPRO-G: an autonomous mobile robot guide for campus using IoT," *International Journal of Trend in Scientific Research and Development*, vol. 2, no. 3, pp. 896–901, 2018.
- [3] X. Liu, J. Yin, L. Jia et al., "Time-efficient tag searching in large-scale RFID systems: a compact exclusive validation method," *IEEE Transactions on Mobile Computing*, vol. 21, no. 4, pp. 2891–2905, 2022.
- [4] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.
- [5] T. A. Ahanger and A. Aljumah, "Internet of things: A comprehensive study of security issues and defense mechanisms," *IEEE Access*, vol. 7, pp. 11020–11028, 2019.
- [6] S. Abdalla Hamad, W. E. Zhang, Q. Z. Sheng et al., "Iot device identification via network-flow based fingerprinting and learning," in *Proceedings of the 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 103–111, IEEE, Rotorua, New Zealand, August 2019.
- [7] S. Marchal, M. Miettinen, T. D. Nguyen, A.-R. Sadeghi, and N. Asokan, "AuDI: toward autonomous IoT device-type identification using periodic communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.
- [8] A. Hameed and A. Leivadeas, "Iot traffic multi-classification using network and statistical features in a smart environment," in *Proceedings of the 25th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 1–7, IEEE, Pisa, Italy, September 2020.
- [9] V. Thangavelu, D. Divakaran, R. Sairam et al., "Deft: a distributed iot fingerprinting technique," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 940–952, 2018.
- [10] F. Yin, Li Yang, Y. Wang, and J. Dai, "Iot etei: end-to-end iot device identification method," in *Proceedings of the IEEE Conference on Dependable and Secure Computing (DSC)*,

- pp. 1–8, IEEE, Aizuwakamatsu, Fukushima, Japan, February 2021.
- [11] Y. Qin, W. Wang, S. Zhang, and K. Chen, “An exploit kits detection approach based on http message graph,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3387–3400, 2021.
 - [12] E. Anthi, L. Williams, M. Slowinska, G. Theodorakopoulos, and P. Burnap, “A supervised intrusion detection system for smart home iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.
 - [13] T. Li, Z. Hong, and Li Yu, “Machine learning-based intrusion detection for iot devices in smart home,” in *Proceedings of the 16th IEEE International Conference on Control & Automation (ICCA)*, pp. 277–282, IEEE, Singapore, October 2020.
 - [14] A. Hamza, H. Habibi Gharakheili, T. A. Benson et al., “Detecting volumetric attacks on iot devices via sdn-based monitoring of mud activity,” in *Proceedings of the ACM Symposium on SDN Research (SOSR)*, pp. 36–48, IEEE, Budapest, Hungary, April 2019.
 - [15] M. Yair, M. Bohadana, Y. Mathov et al., “N-baiot - network-based detection of iot botnet attacks using deep autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.
 - [16] Y. Wan, K. Xu, G. Xue et al., “Iotargos: a multi-layer security monitoring system for internet-of-things in smart homes,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pp. 874–883, IEEE, Toronto, ON, Canada, July 2020.
 - [17] K. Yang, Q. Li, and L. Sun, “Towards automatic fingerprinting of iot devices in the cyberspace,” *Computer Networks*, vol. 148, pp. 318–327, 2019.
 - [18] X. Feng, Q. Li, H. Wang et al., “Acquisitional rule-based engine for discovering internet-of-things devices,” in *Proceedings of the 27th USENIX Security Symposium*, pp. 327–334, USENIX, Baltimore, MD, USA, August 2018.
 - [19] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah, “Gtid: a technique for physical device and device type fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 519–532, 2015.
 - [20] S. Aneja, N. Aneja, and Md S. Islam, “Iot device fingerprint using deep learning,” in *Proceedings of the IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, pp. 174–179, IEEE, Bali, Indonesia, November 2018.
 - [21] N. Msadek, R. Soua, and T. Engel, “Iot device fingerprinting: machine learning based encrypted traffic analysis,” in *Proceedings of the IEEE wireless communications and networking conference (WCNC)*, pp. 1–8, IEEE, Marrakesh, Morocco, April 2019.
 - [22] A. J. Pinheiro, J. Bezerra, C. A. P. Burgardt, and D. R. Campelo, “Identifying iot devices and events based on packet length from encrypted traffic,” *Computer Communications*, vol. 144, pp. 8–17, 2019.
 - [23] J. Frahim, C. Pignataro, J. Aparcar et al., “Securing the internet of things: a proposed framework,” in *White Paper*, CISCO, San Jose, CA, USA, 2015.
 - [24] T. Yu, V. Sekar, S. Srinivasan et al., “Handling a trillion (unfixable) flaws on a billion devices: rethinking network security for the internet-of-things,” in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pp. 1–7, ACM, Baltimore, MD, USA, November 2015.
 - [25] M. Antonakakis, Tim April, M. Bailey et al., “Understanding the mirai botnet,” in *Proceedings of the 26th USENIX security symposium*, pp. 1093–1110, USENIX, Vancouver, BC, Canada, August 2017.
 - [26] G. Kambourakis, C. Kolias, and Angelos Stavrou, “The mirai botnet and the iot zombie armies,” in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pp. 267–272, IEEE, Baltimore, MD, USA, October 2017.
 - [27] Thien Duc Nguyen, S. Marchal, M. Miettinen et al., “Diot: a federated self-learning anomaly detection system for iot,” in *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 756–767, IEEE, Dallas, TX, USA, July 2019.
 - [28] Y. Wan, K. Xu, F. Wang et al., “Characterizing and mining traffic patterns of iot devices in edge networks,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 89–101, 2020.
 - [29] T.-Yu Lin, J.-P. Wu, P.-H. Hung et al., “Mitigating syn flooding attack and arp spoofing in sdn data plane,” in *Proceedings of the 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 114–119, IEEE, Daegu, South Korea, September 2020.
 - [30] J.-S. Park and M.-S. Kim, “Design and implementation of an snmp-based traffic flooding attack detection system,” in *Proceedings of the 11th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 380–389, Springer-Verlag, Berlin Heidelberg.
 - [31] N. Ugtakbayar, D. Battulga, and Sh Sodbileg, “Classification of artificial intelligence ids for smurf attack,” *International Journal of Artificial Intelligence & Applications*, vol. 3, no. 1, pp. 47–51, 2012.
 - [32] M. Yair, M. Bohadana, A. Shabtai et al., “Detection of unauthorized iot devices using machine learning techniques,” *Machine Learning for IoT Security Analytics*, vol. 1709, p. 4647, 2017.
 - [33] J. Sun, K. Sun, and C. Shenefiel, “Automated iot device fingerprinting through encrypted stream classification,” *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer-Verlag, in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 147–167, U.S. ARO grant, December 2019.
 - [34] R. S. Mustafizur, G. Blanc, Z. Zhang et al., “Iot devices recognition through network traffic analysis,” in *Proceedings of the IEEE International Conference on Big Data*, pp. 5187–5192, IEEE, Seattle, WA, USA, December 2018.
 - [35] F. Zhou, W. Huang, Y. Zhao, Y. Shi, X. Liang, and X. Fan, “Entvis: a visual analytic tool for entropy-based network traffic anomaly detection,” *IEEE computer graphics and applications*, vol. 35, no. 6, pp. 42–50, 2015.

Research Article

Destroy the Robust Commercial Watermark via Deep Convolutional Encoder-Decoder Network

Wei Jia,¹ Zhiying Zhu ,² and Huaqi Wang³

¹School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China

²School of Computer Science, Fudan University, Shanghai 200433, China

³School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China

Correspondence should be addressed to Zhiying Zhu; zyzhu19@fudan.edu.cn

Received 6 September 2021; Accepted 10 November 2021; Published 6 December 2021

Academic Editor: Weiwei Liu

Copyright © 2021 Wei Jia et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Nowadays, robust watermark is widely used to protect the copyright of multimedia. Robustness is the most important ability for watermark in application. Since the watermark attacking algorithm is a good way to promote the development of robust watermark, we proposed a new method focused on destroying the commercial watermark. At first, decorrelation and desynchronization are used as the preprocessing method. Considering that the train set of thousands of watermarked images is hard to get, we further use the Bernoulli sampling and dropout in network to achieve the training instance extension. The experiments show that the proposed network can effectively remove the commercial watermark. Meanwhile, the processed image can result in good quality that is almost as good as the original image.

1. Introduction

With the development of Internet technology and smart-phones, the copyright protection of digital images has become increasingly important. Digital watermarking technology is an important branch of information hiding, and it provides a solution for the copyright protection of multimedia products. Watermark can be classified into visible watermarks and invisible watermarks in terms of visibility. A common commercial watermark is visualized. The classic style is a logo with a degree of transparency. The emphasis is on copyright declaration, but it is not safe. Infringement can be reached directly by intercepting or erasing by image processing software. Invisible watermark can be subdivided into robust watermark, semi-fragile watermark, and fragile watermark according to the difference in robustness. Images containing fragile watermark can be easily located after being tampered with, while semi-fragile watermark is robust to certain attacks and only vulnerable to certain specific attacks. Robust watermark is the most widely studied and used watermark. As it is most resistant to attacks, the robust watermark can be extracted

after many kinds of attacks, so it is often used in OSN for copyright protection Voloshynovskiy et al. [1]. The QIM Chen and Wornell [2] algorithm quantifies the original cover into several different index intervals by different quantifiers, which is also the embedding process. The watermarking will be extracted according to the quantitative index interval of modulated data. The receiver can detect hidden data by the shortest distance method when the channel interference is not serious. The spread spectrum code with pseudorandom and cross-correlation properties plays a key role in SS Dixon [3] algorithm, and the energy distribution of embedded watermarking signal is extended to a wider spectrum, which improves the security and robustness capability. ULPM Kang et al. [4] eliminates the interpolation distortion and expands the embedding space. A discrete log-polar point can be obtained by performing the ULPM to the frequency index in the Cartesian system, and the data of which are then embedded to the corresponding DFT coefficient in the Cartesian system. Figure 1 shows the general steps of using the robust watermark in OSN. After transmission through the lossy channel, the robust watermark can still be extracted correctly to protect the copyright.

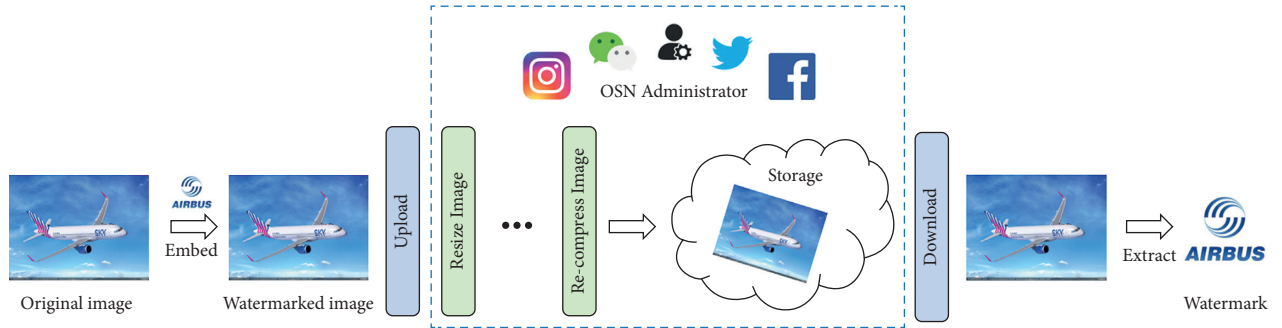


FIGURE 1: Robust watermark used in OSN. “...” used in the image represents the unknown operations.

The original robust watermarking technology was also developed from least significant bit (LSB). In addition, the method based on image pixel histogram is also a representative research in the early stage Coltuc and Bolon [5]. Same as steganography, the embedding domain of image watermark is also divided into spatial domain and transform domain. Compared with embedded in spatial domain, watermark embedded in transform domain usually has less impact on image vision with the same robustness. In the transform domain, the discrete cosine transform (DCT) attracts the most research attention, which is mainly due to the widespread use of JPEG format images Parah et al. [6]. In addition to DCT, the commonly used transform domains include Fourier–Mellin transform (FMT), singular value decomposition (SVD), and discrete wavelet transform (DWT) Li et al. [7]. In practical applications, watermark needs to consider the application scenarios using different transform domains or composite combinations. In addition to the research in laboratories, the commercial watermark Digimarc is also very typical. This application is integrated with software Adobe Photoshop in the form of a plug-in.

Unlike steganography, which hardly considers active attacks, watermark faces a variety of attacks. Images transmitted in OSN have their compression operation. Image cropping, video capture tools, and the addition of mosaics and textures are also very common. In terms of types, anti-watermarking can be roughly divided into removal attacks Su and Girod [8]; geometric attacks D’angelo et al. [9] D’Angelo [10]; cryptographic attacks Cox and Linnartz [11]; and protocol attacks Craver et al. [12]; Kutter et al. [13]. Cryptographic attacks and protocol attacks are mainly aimed at watermarks that use cryptographic theories, and the hidden vulnerabilities in the transmission of watermarking images. Due to limited applications, they have been rarely studied nowadays. However, removal attacks and geometric attacks can still be seen in cutting-edge watermarking research. The most common one is JPEG compression. After the lossy compression process, the watermarked image loses its information, and the amount of loss is related to the compression strength. The geometric attack is to geometrically warp the image to change the position of the original pixel coefficients and break the rules of the watermarking algorithm.

In the field of information security, we often research the two sides of the problem. The research of new watermarking

attacks is not only to put forward better standards for measuring the robustness of watermarks but also to prevent watermarking algorithms from being applied to illegal transmission. We need better watermarking attack technology, which can destroy the watermark more effectively than the traditional methods while ensuring that the quality of the processed image will not be affected too much. In today’s popularization of deep learning research, we should consider using new related technologies to update methods.

In this study, we proposed a new watermarking attack framework focused on destroying the commercial watermark. The advantages of our proposed method lie in the following twofold:

- (i) We proposed a preprocessing method, which includes two parts: decorrelation and desynchronization.
- (ii) The results of experiments show the excellent attack ability of our network, and compared with traditional attacks, the processed image maintains good image quality.

The rest of this study is organized as follows. Section 2 reviews the related work with our method. The proposed scheme is specified in detail in Section 3. Section 4 provides the experimental results, and Section 5 concludes the study.

2. Related Work

2.1. Attack Methods on Digital Watermark. The watermarking attack technology has been developed for many years. We mainly introduce two traditional watermark attacking methods: removal attacks and geometric attacks. It is not necessary to know the principle of watermarking algorithm to remove attacks and geometric attacks. Destroying the secret carrier is essentially destroying the watermarking information probabilistically. The increasingly developed watermarking technology has also developed adequate countermeasures.

The removal attack aims to completely remove the watermark from the protection carrier, and it is the most used attack method with the most categories. The removal attack can be divided into denoising attack Shukla et al. [14], remodulation attack, and lossy compression attack Wallace [15], mainly using filtering, coding, and other technologies Langelaar et al. [16]. The basic idea of the denoising attack is

to assume that the watermark is a layer of additive noise in the carrier, which theoretically defines the expected goal of removing the watermark. Considering the speed requirements of real-time attack applications, Geng, Zhang, Chen, Fang and Yu [17] used a denoising network based on DnCNN Zhang et al. [18] architecture to remove watermark. The inputs of the network are the watermarked image preprocessed, and the corresponding original images are as labels. In this network, the residual features extracted by CNN are considered as the watermark, and the time cost of estimate is short enough to meet most real-time requirements and destroys the correlation between the watermarked image and the real watermark, which causes the burden of the watermarking decoder. Lossy compression attacks use the JPEG compression method to compress the three YUV components of the image. Different from the removal attack to eliminate the embedded watermark, the geometric attack method is used usually to spatially warp the carrier image to change the original pixel position. The purpose is to make the watermarking extraction algorithm and the embedded information lose synchronization, to complete the destruction of the watermark.

The other methods of attack include cryptographic attacks and average joint attacks. The attack methods of cryptographic attacks are very similar to the early methods of decrypting passwords, and the calculation is very complicated. A cryptographic attack method can only target the watermark of a specific method. The most representative one is the oracle attack method. Average attacks and joint attacks are mostly used on video watermark Deguillaume et al. [19], Pereira and Pun [20]. Video is composed of continuous images, and there is a strong correlation between frames, especially in static scenes. Both the average and joint attack operations require the use of a large number of video frames as the datasets.

According to Zhu, Kaplan, Johnson, and Fei-Fei [21], a triple convolutional network of encoder, decoder, and discriminator is used. The carrier information and embedded information are input into the encoder to obtain the coded image. The decoder is responsible for reconstructing information from the coded image, and the discriminator guarantees the quality of the watermarked image. The coded image is attacked, so that the decoder can extract watermarking information from the attacked image and realize the robustness of the watermark. This method is stable and excellent in the face of various traditional attack methods. We can also realize that the research on anti-watermarking is far from enough in this study. We should consider more comprehensive and complete attack methods to promote the development of watermark.

2.2. Digimarc Watermark. Digimarc is a commercial watermark, usually used as a plug-in integrated into the application software, the most common of which is Adobe Photoshop. Like the watermarks mentioned in the introduction, the Digimarc watermark is visually invisible with strong robustness. When embedding a watermark, the available options include image information, image

attributes, and watermarking durability. The first two are differences in information content, and watermarking durability is the most critical option. The software divides the durability into four levels. The levels are denoted by a in the experimental part below. As the number increases, the embedded watermark is more robust, and the corresponding impact on the vision of cover is also increased. When extracting information, the software will display two results, one is the content of the embedded information and the other is the strength of the extracted watermark. The strength of the watermark cannot be quantified, but it can be roughly divided into six levels, namely very strong, strong, medium, weak, very weak, and none. If it is only the weakening of the intensity, the expected purpose cannot be achieved. Only when the result is displayed as none, the watermark is considered to be destroyed.

3. Proposed Framework

Different from the watermarking algorithms studied in laboratory, as a commercial watermark, the principle of Digimarc is unknown. Therefore, we cannot design and optimize the network in a targeted manner according to the selection of their embedding domain and the process of embedding and extracting information, such as the use of high-pass filters. Used as a plug-in, the Digimarc watermark is a complete black box, and it is difficult to have prior knowledge that can be relied on.

Figure 2 shows the framework of Digimarc watermark attacking model. To ensure the success of the attack, the input watermarked image is preprocessed before the training starts. The preprocessing includes two operations of decorrelation and synchronization. In addition, different from the datasets usually used in network training, it is hard to get thousands of images with the same type of commercial watermark. Therefore, we use a deep convolutional encoder-decoder network based on single-image training. The network is based on the idea of removing attacks, regarding the Digimarc watermark as additive noise on the cover, and allows the original image to be used as a learning target. The main body is made up of an encoder-decoder network. To fully avoid the overfitting of small data training, some neurons are dropped out during the training and testing phases, and the Bernoulli rules are followed. In the early stage of training, the train set is expanded by the Bernoulli sampling to further improve the generalization ability.

3.1. The Preprocessing of Watermarked Images. Before training the encoder-decoder network, we first proposed a preprocessing method. The preprocessing of the input image mainly includes two parts: decorrelation and desynchronization. The flow chart is shown in Figure 3. Assuming that the length and width of the watermarked image I_W are H and W , respectively, we applied a Wiener filter with $\varphi \times \varphi$ size filter kernel in decorrelation. Starting from minimizing the mean square error, the purpose is to reduce the correlation between the watermark signal and the original carrier. Assuming that the original carrier image is I_C and

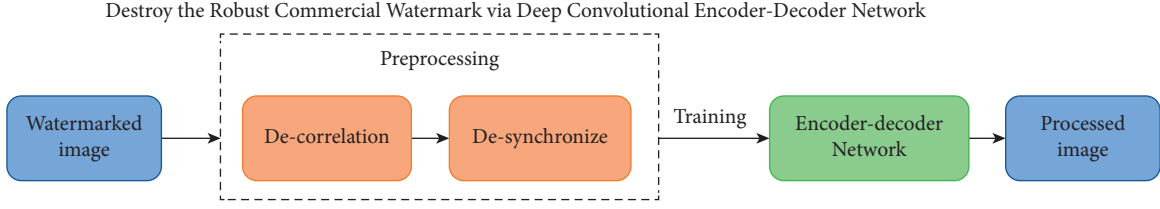


FIGURE 2: The framework of Digimarc watermark attacking model.

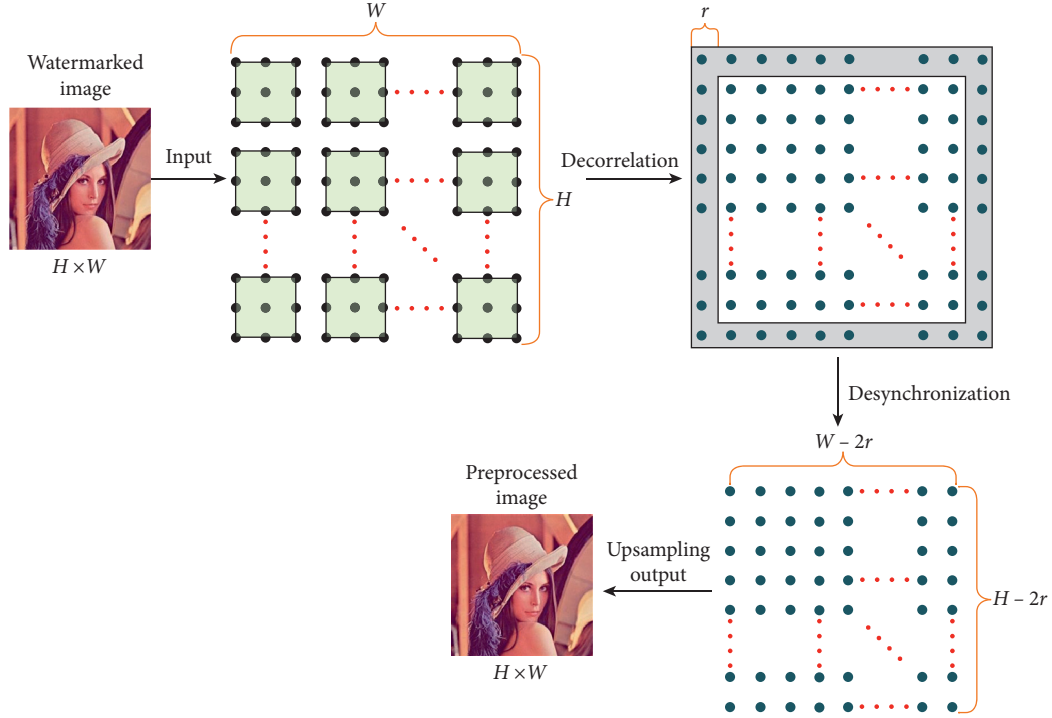


FIGURE 3: Preprocessing operations on watermarked images.

the reconstructed image after decorrelation is I_M , the mean square error between the two images can be expressed as follows:

$$\begin{aligned} \text{MSE} &= E[I_C(n) - I_M(n)]^2 \\ &= E[I_C^2(n)] - 2E[I_C(n)I_M(n)] + E[I_M^2(n)], \end{aligned} \quad (1)$$

where $E[\cdot]$ represents mathematical expectation, $n = 1, 2, \dots, H \times W$. The right part of equation (1) can be further derived as follows:

$$\begin{aligned} E[I_C^2(n)] &= R_{I_C}(0), \\ E[I_C(n)I_M(n)] &= E\left[I_C(n) \sum_{k=-\infty}^{\infty} h(k)I_M(n-k)\right] \\ &= \sum_{k=-\infty}^{\infty} h(k)R_{I_C I_W}(k), \\ E[I_M^2(n)] &= \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(k)h(m)R_{I_W}(m-k). \end{aligned} \quad (2)$$

Among them, $R(\cdot)$ represents the correlation function, and h is the filter element. If h_0 is the optimal filter that meets the minimum mean square error, then h can be expressed as follows:

$$h(n) = h_0(n) + g(n). \quad (3)$$

Among them, g is the error. Substituting h_0 into equation (1), we can get the minimum mean square error 0. Combining the above equation, we can get the following:

$$\begin{aligned} \text{MSE} &= \text{MSE}_0 + f_1 + f_2, \\ f_1 &= 2 \sum_{m=-\infty}^{\infty} g(m) \left[\sum_{k=-\infty}^{\infty} h_0(k)R_{I_W}(m-k) - R_{I_C I_W}(m) \right], \\ f_2 &= \sum_{m=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} g(m)g(k)R_{I_W}(m-k). \end{aligned} \quad (4)$$

Since MSE must be greater than or equal to MSE_0 , it is easy to know that f_2 must be greater than zero, so f_1 needs to be equal to zero.

$$R_{I_C I_W}(m) = \sum_{k=-\infty}^{\infty} h_0(k) R_{I_W}(m-k). \quad (5)$$

Converting equation (5) into the form of power spectrum, the filter expression at the energy level can be obtained as follows:

$$H_0(s) = \frac{\Phi I_C I_W(s)}{\Phi I_W(s)}. \quad (6)$$

Equation (6) is the key to reducing the correlation between the watermark and the original carrier. To explain from the perspective of image pixels, the local neighborhood of the pixel can be used to estimate the mean and standard deviation of the filter h under the filter kernel size φ .

$$\begin{aligned} \mu &= \frac{1}{\varphi^2} \sum_{i,j \in \eta} b_{i,j}, \\ \sigma &= \sqrt{\frac{1}{\varphi^2} \sum_{i,j \in \eta} b_{i,j}^2 - \mu^2}. \end{aligned} \quad (7)$$

Among them, $b_{i,j}$ is the (i, j) pixel in I_W , so the decorrelated image I_M can be expressed as follows:

$$I_M = I_W \otimes H. \quad (8)$$

Among them, \otimes represents the convolution operation, and H is the convolution kernel of the filter. The filter core of $\varphi = 3$ is used in this scheme. The neighborhood of pixel 8 in the natural image has the strongest correlation. Choosing this size is enough to complete the initial weakening of the correlation between the watermark and the carrier image. In Figure 3, the filter kernel is represented as a green square. After convolving the black pixels of I_W , the blue pixels in I_M are obtained.

Further, the image I_M is desynchronized, the pixel collection matrix in I_M by M is indicated, and the pixel matrix of the image I_T after desynchronization can be expressed as follows:

$$T = M\{r: W - r, r: H - r\}. \quad (9)$$

We set $r = 1$ here, and finally, I_T will be upsampled to the original image size to complete the preprocessing of the watermarked image.

3.2. Data Augmentation. Many image processing methods, such as denoising, restoration, and super-resolution, often have a similar goal, which is to minimize the difference between the image generated and the original image. The goal can usually be expressed by the following equation:

$$\tilde{y} = \min_y E(y; x) + R(y), \quad (10)$$

where y represents the generated image and x represents the original image. $E(y; x)$ is an optimization goal, which will be changed according to different requirements. The most common optimization goal is $\|y - x\|^2$. $R(y)$ is a regularizer, which is generally obtained as a priori information based on

a large amount of dataset. The choice of regularizer, which usually captures a generic prior on natural images, is more difficult and is the subject of much research. In this work, we replace the regularizer $R(y)$ with the implicit prior captured by the neural network as follows:

$$\begin{aligned} \theta^* &= \arg \min_{\theta} E(f_{\theta}(z); x), \\ \tilde{y} &= f_{\theta^*}(z). \end{aligned} \quad (11)$$

The minimizer θ^* is obtained using an optimizer such as gradient descent starting from a random initialization of the parameters.

In DIP Ulyanov et al. [22], the author believes that the same training results can be obtained by relying on the parameters of the network itself through only one image used for repeated iterations. DIP proposed a hand-designed priori function. In some cases, the performance is comparable to that of networks based on large datasets. The most urgent problem for single-image training is overfitting. If the network is regarded as the Bayesian estimation, then the prediction accuracy can be represented by the mean square error (MSE). The variance in the MSE will increase sharply because the training samples are particularly small, which will make the model lose its generalization ability. Therefore, the focus of network design is to reduce the variance to solve the problem of overfitting.

While learning only by a single image, to make full use of the information of the train sets, we first extended with the single image. We use the Bernoulli sampling to generate a large number of samples. These samples essentially contain image content, but are different from the original image, and have a good optimization effect on training tasks. The Bernoulli distribution can be explained simply by the coin tossing problem, and it describes a binary problem. For example, for a variable x , there are two possible values of 0 and 1, and the probability of $x = 1$ can be represented in formula (12). When expressed as the Bernoulli distribution, it can be written as formula (13).

$$P(x = 1 | \rho) = \rho (0 \leq \rho \leq 1), \quad (12)$$

$$P(x | \rho) = (1 - \rho)^{1-x} \rho^x. \quad (13)$$

For a sample set, although the samples are independent of each other, but conform to the same distribution, the likelihood function can be expressed by the following formula:

$$\begin{aligned} P(S | \rho) &= \prod_{m=1}^M P(x_m | \rho) \\ &= \prod_{m=1}^M (1 - \rho)^{1-x_m} \rho^{x_m}, \end{aligned} \quad (14)$$

where S represents the sample set, $S = \{x_1, x_2, \dots, x_m\}$.

In the field of image computing, it can be represented by a simple mathematical model. In this study, we suppose that the original image and the watermarked image are I_C and I_W , respectively, and expanded into corresponding sample pairs with a total of M copies. The sample pairs can be represented as $\{I_C^m\}_{m=1}^M$ and $\{I_W^m\}_{m=1}^M$.

$$\{\tilde{I}_{\text{label}}^m\}_{m=1}^M = B_n \odot I_{\text{label} m=1}^M, \quad (15)$$

where label represents C or W . In practical applications, P_m is the Bernoulli distribution with the same size as the tensor and the probability conforms to ρ . Denote the element as K , and (15) can be redefined as follows:

$$\tilde{I}_{\text{label}}^m[k] = \begin{cases} I_{\text{label}}[k], & \rho, \\ 0, & 1 - \rho. \end{cases} \quad (16)$$

Let $g_\theta(\cdot)$ be the objective mapping function, and then, the loss function can be expressed as follows:

$$\min_\theta \sum_{m=1}^M \left\| g_\theta(\tilde{I}_W^m) - \tilde{I}_C \right\|_{B_n}^2. \quad (17)$$

In this subsection, we use the Bernoulli sampling to increase the training samples and initially solve the overfitting problem. The loss of each pair of samples is only performed on the mask represented by the current Bernoulli distribution. Because the mask tensor is completely selected at random, when we obtain a sufficient number, we can use the sum of the loss of all samples to measure the perceptual loss on the overall image.

3.3. Network Architecture. Inspired by Quan, Chen, Pang, and Ji [23], we propose to use the network depicted in Figure 4. This model has been proved useful in denoising task. Different from the existing denoising methods, this network can only use the input noisy image itself for training, which meets our need perfectly. It is mainly composed of two parts: encoder and decoder. Assuming that the original image I_C is embedded in the Digimarc watermark by software, the watermarked image I_W is obtained. The size of each image is $H \times W \times C$, H and W represent the length and width of the image, and C is the number of image channels. To prevent overfitting, before I_W is sent to the network for training, it is first sampled by the Bernoulli sampling to get the sample set. Then, after a partial convolution (PConv, partial convolutional), and activated by LeakyReLU, 3×3 PConv is used for pixel normalization.

The whole encoder is composed of six coding units. The first five coding units have a similar structure, and they are all composed of a local convolution layer activated by LeakyReLU and a maximum pooling layer with a sampling kernel of 2×2 size with the step of 2. The feature number of each layer is set to 64, and finally, the result is output by the last coding unit, and then, it is upsampled to the decoder.

The decoder contains a total of five decoding units, the first four are collectively referred to as decoding unit A and the last one is decoding unit B . For each decoding unit, the encoding feature map is connected first, and after that, two 3×3 convolution layers are applied. The convolutional layers are also activated by LeakyReLU. To further solve the overfitting problem, “dropout” is applied to delete some neurons and then continues to enter the next decoding unit through an upsampling. The numbers of convolution

kernels of the decoding unit A are all 128, and the number of channels of the convolutional layer included in the decoding unit B is 96, 64, 32, and C , and finally, the output image is generated.

The convergence goal of the network is given in formula (17) in the previous section. As can be seen from the introduction in this section, the single-image training network must first face the problem of its own overfitting and then the performance index. Although it is difficult to achieve the performance of a network based on large datasets by learning from a single image under the full conditions, it can play a very good effect when the data source is lacking.

4. Experimental Results and Analysis

4.1. The Method of Test. Assuming that $g_\theta(\cdot)$ is the mapping function model obtained by training, if the test image is processed by this model, it is equivalent to a mapping of the sampled instance from test set. Generally, a network with “dropout” will scale the weight of the model through related rules during testing. Therefore, to better optimize the performance, in the actual testing phase, we use the Bernoulli sampling and “dropout” again on the model to generate a set of submodels and then average the final results. We define the submodel set of $g_\theta(\cdot)$ after K Bernoulli sampling as $\{g_\theta^k(\cdot)\}_{k=1}^K$ and the generated image I_G can be expressed as follows:

$$I_G = \frac{1}{K} \sum_{k=1}^K g_\theta^k(B_{N+k} \cdot I_{\text{Pre}}). \quad (18)$$

4.2. Experimental Settings. In this section, we conduct experiments to analyze the performance of proposed network. The goal of this study is the commercial Digimarc watermark, and the embedding and extraction of the watermark are carried out on the Photoshop software platform. The datasets we use are Set9 and Set14 Bevilacqua et al. [24]; all images are color and resized as 512×512 . The examples of test image are shown in Figure 5. As mentioned in Subsection 2.2, the most important variable in the embedding process is the durability of the watermark, where the variable S refers to the strength of the watermark. Each group of experiments will test the watermarked images of four intensities separately. The objective evaluation indicators include the strength of the watermark after attack, PSNR, and SSIM. PSNR and SSIM are two common metrics for assessing the quality of the reconstructed image. Higher PSNR and SSIM values generally indicate a better quality of the image Hore and Ziou [25]. We therefore used the average PSNR and SSIM values to assess the quality of processed images. In addition, to better reflect the excellent performance of the proposed network, we have also compared the proposed method with StirMark. The most famous software in the traditional anti-watermarking technology is StirMark software. The integration of this application covers almost all traditional watermark attacking methods, which can be described as a benchmark. Because of the variety of

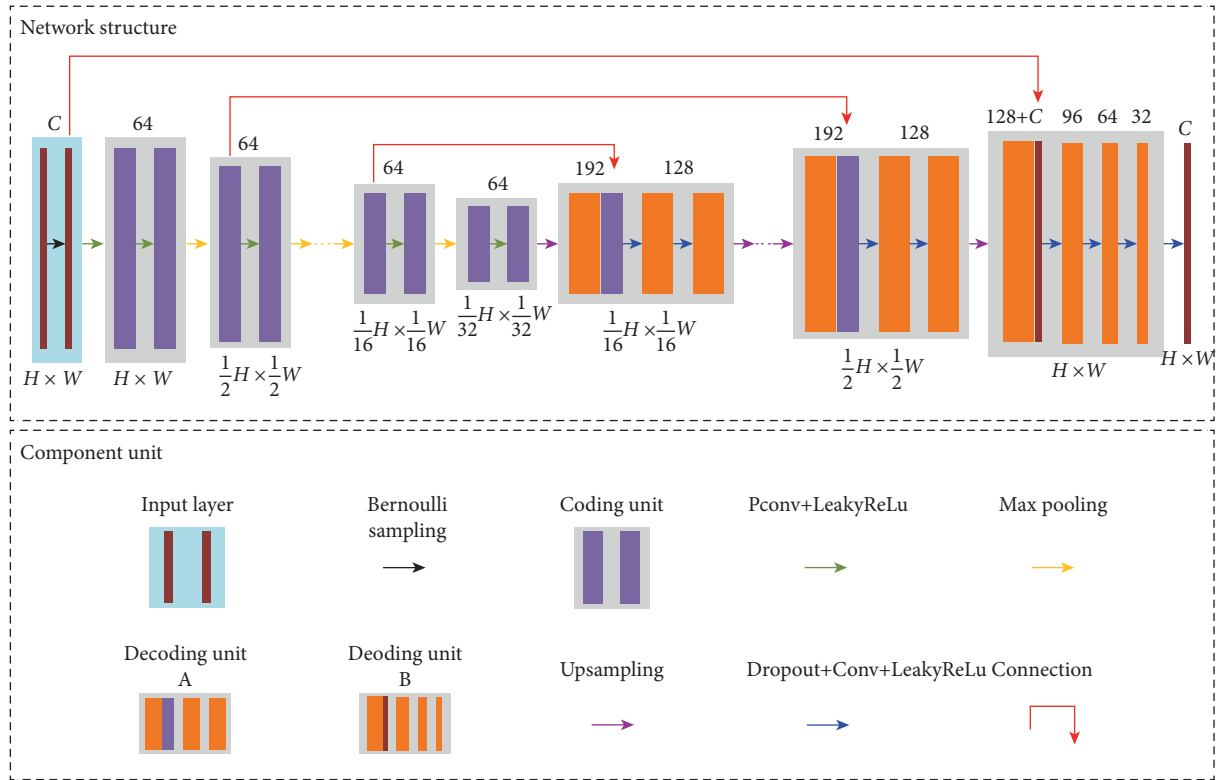


FIGURE 4: Overview of the network architecture.

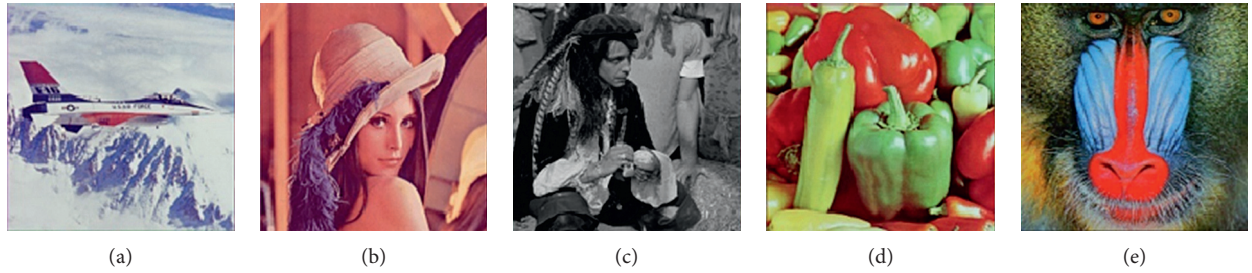


FIGURE 5: The examples of test image. (a) F16. (b) Lena. (c) Man. (d) Pepper. (e) Baboon.

TABLE 1: Strength of watermark extracted after different methods of attack.

Test images		$F16_{a1}$	$F16_{a2}$	$F16_{a3}$	$F16_{a4}$	
Proposed method		None	None	None	None	
Wang et al. [26]		None	None	None	None	
StirMark	Cropping	$c = 0.75$	Weak	Medium	Medium	Medium
		$c = 0.5$	Very weak	Weak	Weak	Weak
		$c = 0.25$	None	None	None	None
	Rotation	$r = 180^\circ$	Medium	Medium	Medium	Medium
		$r = 150^\circ$	Medium	Medium	Medium	Medium
		$r = 90^\circ$	Medium	Medium	Medium	Medium
	Scaling	$sc = 0.5$	Weak	Weak	Weak	Weak
$sc = 1.5$		Medium	Strong	Strong	Strong	
	$sc = 2$	Medium	Medium	Medium	Medium	

TABLE 2: Visual effects after being processed by various attacks. The strength of extracted watermark is in the ().


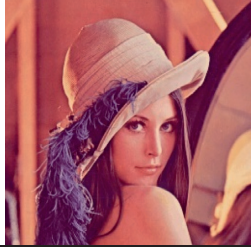

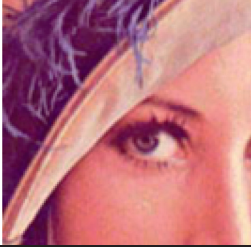


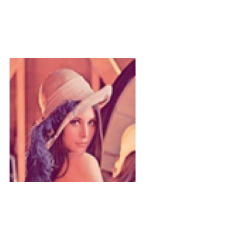
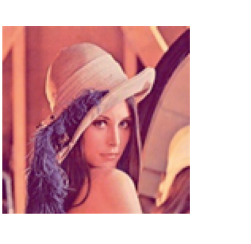







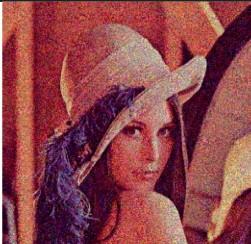
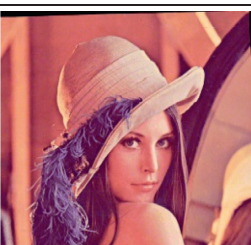
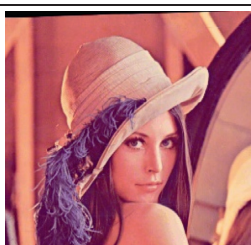
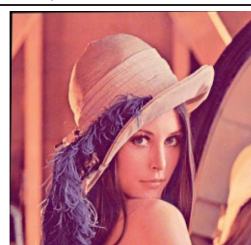

Lena_a3	The proposed method	Cropping $c=0.75$	Cropping $c=0.25$
			
(None)	(None)	(Medium)	(None)
Rotation $r=180$	Rotation $r=90$	Scaling $sc=0.5$	Scaling $sc=0.75$
(Medium)	(Medium)	(Very weak)	(Medium)
			
JPEG Compress $QF=50$	JPEG Compress $QF=30$	Desynchronization $SS=3$	Desynchronization $SS=2$
(Weak)	(Very weak)	(Strong)	(Medium)
			
Median filter $k=5$	Median filter $k=9$	Gaussian noise $=4$	Gaussian noise $=8$
(Strong)	(Weak)	(Weak)	(Very weak)
			
Random distortion $d=0.95$	Random distortion $d=1.05$	Affine transformation $f=5$	Affine transformation $f=2$
(Medium)	(Medium)	(Very weak)	(Weak)
			

TABLE 3: Performance comparison in terms of removal effect and image quality for test images.

Test images	Man _{a4}			Pepper _{a4}			Baboon _{a4}			
	Strength of watermark	PSNR	SSIM	Strength of watermark	PSNR	SSIM	Strength of watermark	PSNR	SSIM	
Proposed method	None	39.48	0.9744	None	36.76	0.9154	None	36.35	0.9413	
JPEG	Q = 30	None	24.28	0.7866	Very weak	23.04	0.7627	Very weak	22.96	0.7805
	Q = 50	Very weak	24.76	0.8248	Weak	24.45	0.8002	Weak	23.61	0.8275
	Q = 70	Weak	25.18	0.8604	Weak	24.76	0.8387	Weak	24.13	0.8626
Gaussian noise	$\sigma = 2$	Medium	26.61	0.6703	Medium	26.50	0.5417	Medium	26.48	0.7771
	$\sigma = 4$	Weak	20.94	0.4308	Weak	20.61	0.2867	Weak	20.51	0.5351
	$\sigma = 8$	Very weak	14.83	0.2171	Very weak	14.49	0.1293	Weak	14.55	0.2768
Median filter	$k = 2$	Strong	22.77	0.7799	Medium	23.55	0.8760	Medium	21.40	0.7797
	$k = 5$	Strong	22.59	0.6687	Medium	23.58	0.8190	Medium	20.53	0.5483
	$k = 7$	Medium	22.10	0.5944	Medium	23.42	0.7794	Medium	19.92	0.4225

watermark attacking method types, only the representative part is selected in the experiment of this article.

We initial the Adam optimizer with a learning rate of $1e - 4$ to train the model and train our network on a single GPU NVIDIA GTX1080ti for 1.5×10^5 iterations. The time cost of each training is about 3.5 hours. The LeakyReLU activation function with a hyperparameter of 0.1, the Bernoulli with a sampling probability of 0.1, and a decoder with a “dropout” rate of 0.3 are used in training. In the test phase, a total of 100 dropouts were performed on the model, and the actual result is the average of the 100 processing results.

4.3. Performance Evaluation. In this section, we will show the results of attack on watermarking of different strengths through our method and the comparison experiments with Wang, Qian, Feng, and Zhang [26] and StirMark. We choose the “F16” in Set9 as the test image and embed the watermark with the strength a with 1, 2, 3, and 4, and then, we will get the four images to be attacked. Table 1 shows the results after processing with the method proposed, and the comparison with Wang et al. [26] and several common geometric attacks in StirMark. Let the parameter of the cropping attack be c , and the value is $\{0.75, 0.5, 0.25\}$; the rotation attack parameter be r , and the value is $\{180^\circ, 150^\circ, 90^\circ\}$; and the scaling attack parameter be sc , and the value is $\{0.5, 1.5, 2\}$. It can be seen from the table that the Digimarc watermark has a very high correction ability in resisting rotation attacks. In a scaling attack, the reduction operation weakens the watermark more than the zoom operation. The cropping attack has a higher destruction rate than the other two methods, and it can completely eliminate the watermark in the case of great damage to the image. For the Digimarc watermark, the weakening of the strength does not mean that the watermarking information disappears. Only the effect of the attack makes the software identification result “none,” and this attack is meaningful. Although the method proposed in Wang et al. [26] can also meet the requirement, the proposed method is more convenient when training. In our opinion, the visual quality of the image should not be affected too much while the watermark is erased. Different from StirMark, our method can completely erase watermark of different durability while maintaining good image quality.

To show the visual effect of our method more intuitively, we take the Lena image as the test image and embed the Digimarc watermark with intensity 3 in it. Table 2 shows the results of Lena_{a3} under each attack. The quality factor of JPEG compression is defined as QF, the mean value of Gaussian noise is 0, and the standard deviation is defined as σ . The kernel size of the median filter is k . The parameters of desynchronization attack, random distortion attack, and affine transformation are denoted by ss , d , and f , respectively.

It can be seen from Table 2 that although traditional attacks have different methods, the results will inevitably cause serious visual distortion, and the watermark still exists in this case. Taking the most common JPEG compression attack in reality as an example, the image compression factor in online social networks (OSN) will be around 70, and even when the QF is set as 30, the Digimarc watermark cannot be completely removed. The quality of the watermarked image processed by our method is almost unchanged from that before processing. Table 3 used two objective evaluation indicators, PSNR and SSIM, to illustrate the image quality after processing.

In Table 3, three test images of Man, Pepper, and Baboon are selected for watermark embedding. When the embedding intensity is 4, “Man_{a4},” “Pepper_{a4},” and “Baboon_{a4}” are obtained, respectively. The result shows the comparison of the watermarking strength and visual quality of the test image after several methods of attack. Table 3 further verifies the robustness of Digimarc watermark itself. Even if the standard deviation of Gaussian noise is set to 8, the watermark still cannot be removed. As a comparison, our method can not only remove the watermark perfectly, but also improve the image quality compared with other methods. Taking the “Man” as an example, compared with our method, the PSNR is improved by 24.65 dB, and the SSIM is improved by 0.7573. The image quality has reached a level that cannot be detected by the human eye.

We use four intensities to embed watermark on the five test images. After all the objects are processed by the proposed method, no watermark can be extracted from the images. Table 4 also shows the image quality results of all the attacked images, which meets the expectation that the attack is invisible.

TABLE 4: Image quality of watermarked images with different intensities after being attacked.

		F16	Lena	Man	Pepper	Baboon
$a = 1$	PSNR	40.10	39.68	41.29	38.69	37.29
	SSIM	0.956 0	0.953 0	0.982 1	0.934 3	0.946 6
$a = 2$	PSNR	39.36	38.74	40.37	37.75	36.69
	SSIM	0.950 3	0.946 2	0.978 3	0.926 4	0.942 4
$a = 3$	PSNR	38.80	38.01	39.95	37.27	36.48
	SSIM	0.945 6	0.940 7	0.976 6	0.920 9	0.942 6
$a = 4$	PSNR	38.37	37.20	39.48	36.76	36.35
	SSIM	0.941 8	0.932 3	0.974 4	0.915 4	0.941 3

5. Conclusions

In this study, we propose an attack model against the commercial Digimarc watermark. We have solved the problem that the principle of commercial watermark is unknown and the amount of training data is scarce. The attack rate of our method is very high, so that the watermarking information after the attack cannot be extracted completely, not only weakening the strength of the watermark. The experimental results show that our method has a remarkable improvement in attack performance and image quality compared with various attack methods in StirMark. In the future, we will explore how to add the process of extracting the watermark to the back propagation, which may achieve better results.

Data Availability

No data were used in the study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

- [1] S. Voloshynovskiy, S. Pereira, V. Iquise, and T. Pun, "Attack modelling: towards a second generation watermarking benchmark," *Signal Processing*, vol. 81, no. 6, pp. 1177–1214, 2001.
- [2] B. Chen and G. W. Wornell, "Quantization index modulation: a class of provably good methods for digital watermarking and information embedding," *IEEE Transactions on Information Theory*, vol. 47, no. 4, pp. 1423–1443, 2001.
- [3] R. C. Dixon, *Spread Spectrum Systems: With Commercial Applications*, John Wiley & Sons, Hoboken, NJ, USA, 1994.
- [4] X. Kang, J. Huang, and W. Zeng, "Efficient general print-scanning resilient data hiding based on uniform log-polar mapping," *IEEE Transactions on Information Forensics and Security*, vol. 5, pp. 1–12, 2010.
- [5] D. Coltuc and P. Bolon, "Robust watermarking by histogram specification," in *Proceedings of the 1999 International Conference on Image Processing (Cat. 99CH36348)*, pp. 236–239, IEEE, Kobe, Japan, October 1999.
- [6] S. A. Parah, J. A. Sheikh, N. A. Loan, and G. M. Bhat, "Robust and blind watermarking technique in DCT domain using inter-block coefficient differencing," *Digital Signal Processing*, vol. 53, pp. 11–24, 2016.
- [7] C. Li, Z. Zhang, Y. Wang, B. Ma, and D. Huang, "Dither modulation of significant amplitude difference for wavelet based robust watermarking," *Neurocomputing*, vol. 166, pp. 404–415, 2015.
- [8] J. K. Su and B. Girod, "Power-spectrum condition for energy-efficient watermarking," *IEEE Transactions on Multimedia*, vol. 4, no. 4, pp. 551–560, 2002.
- [9] A. D'angelo, M. Barni, and N. Merhav, "Stochastic image warping for improved watermark desynchronization," *EURASIP Journal on Information Security*, vol. 2008, pp. 1–14, Article ID 345184, 2008.
- [10] A. D'Angelo, *Characterization and quality evaluation of geometric distortions in images with application to digital watermarking*, Ph.D. thesis, Springer, Berlin, Germany, 2009.
- [11] I. J. Cox and J. P. M. G. Linnartz, "Some general methods for tampering with watermarks," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 587–593, 1998.
- [12] S. A. Craver, N. D. Memon, B. L. Yeo, and M. M. Yeung, "Can invisible watermarks resolve rightful ownerships?" in *Storage and Retrieval for Image and Video Databases V* International Society for Optics and Photonics, Bellingham, WA, USA, 1997.
- [13] M. Kutter, S. V. Voloshynovskiy, and A. Herrigel, "Watermark copy attack," in *Security and Watermarking of Multimedia Contents II*, pp. 371–380, International Society for Optics and Photonics, Bellingham, WA, USA, 2000.
- [14] A. K. Shukla, R. K. Pandey, S. Yadav, and R. B. Pachori, "Generalized fractional filter-based algorithm for image denoising," *Circuits, Systems, and Signal Processing*, vol. 39, no. 1, pp. 363–390, 2020.
- [15] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.
- [16] G. C. Langelaar, R. L. Lagendijk, and J. Biemond, "Removing spatial spread spectrum watermarks by non-linear filtering," in *Proceedings of the 9th European Signal Processing Conference (EUSIPCO 1998)*, pp. 1–4, IEEE, Rhodes, Greece, September 1998.
- [17] L. Geng, W. Zhang, H. Chen, H. Fang, and N. Yu, "Real-time attacks on robust watermarking tools in the wild by CNN," *Journal of Real-Time Image Processing*, vol. 17, pp. 1–11, 2020.
- [18] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.
- [19] F. Deguillaume, G. Csurka, and T. Pun, "Countermeasures for unintentional and intentional video watermarking attacks," in *Security and Watermarking of Multimedia Contents II*, pp. 346–357, International Society for Optics and Photonics, Bellingham, WA, USA, 2000.

- [20] S. Pereira and T. Pun, "Fast robust template matching for affine resistant image watermarks," in *Proceedings of the International Workshop on Information Hiding*, pp. 199–210, Springer, Cambridge, UK, May 1999.
- [21] J. Zhu, R. Kaplan, J. Johnson, and L. F. Fei, "Hidden: hiding data with deep networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 657–672, Munich, Germany, September 2018.
- [22] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454, Salt Lake City, UT, USA, June 2018.
- [23] Y. Quan, M. Chen, T. Pang, and H. Ji, "Self2self with dropout: learning self-supervised denoising from single image," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1890–1898, IEEE, Seattle, WA, USA, June 2020.
- [24] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. A. Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proceedings of the British Machine Vision Conference (BMVC)*, Guildford, England, September 2012.
- [25] A. Hore and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *Proceedings of the 2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, IEEE, Istanbul, Turkey, August. 2010.
- [26] H. Wang, Z. Qian, G. Feng, and X. Zhang, "Defeating data hiding in social networks using generative adversarial network," *EURASIP Journal on Image and Video Processing*, vol. 2020, pp. 1–13, 2020.

Research Article

LightSEEN: Real-Time Unknown Traffic Discovery via Lightweight Siamese Networks

Ji Li ¹, Chunxiang Gu ^{1,2}, Fushan Wei ¹, Xieli Zhang¹, Xinyi Hu ¹, Jiaxing Guo ¹,
and Wenfen Liu ³

¹State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China

²Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450001, China

³School of Computer Science and Information Security, Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin, Guangxi 541004, China

Correspondence should be addressed to Chunxiang Gu; gcx5209@126.com

Received 2 July 2021; Accepted 30 September 2021; Published 18 October 2021

Academic Editor: Weiwei Liu

Copyright © 2021 Ji Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the increase in the proportion of encrypted network traffic, encrypted traffic identification (ETI) is becoming a critical research topic for network management and security. At present, ETI under closed world assumption has been adequately studied. However, when the models are applied to the realistic environment, they will face unknown traffic identification challenges and model efficiency requirements. Considering these problems, in this paper, we propose a lightweight unknown traffic discovery model LightSEEN for open-world traffic classification and model update under practical conditions. The overall structure of LightSEEN is based on the Siamese network, which takes three simplified packet feature vectors as input on one side, uses the multihead attention mechanism to parallelly capture the interactions among packets, and adopts techniques including 1D-CNN and ResNet to promote the extraction of deep-level flow features and the convergence speed of the network. The effectiveness and efficiency of the proposed model are evaluated on two public data sets. The results show that the effectiveness of LightSEEN is overall at the same level as the state-of-the-art method and LightSEEN has even better true detection rate, but the parameter used in LightSEEN is 0.51% of the baseline and its average training time is 37.9% of the baseline.

1. Introduction

Network traffic identification refers to classifying network traffic into different sets by observing its characteristics according to specific targets, which is the focus of network behaviour analysis, network planning and construction, network anomaly detection, and network traffic model research [1]. In recent years, with the rapid development of network technology and the widespread use of encryption technology in the network, the amount of encrypted network traffic has gained a fierce increase, and the issue of encrypted traffic identification (ETI) has attracted wide attention from researchers.

Currently, ETI in closed environments has been amply studied. However, for the application in an open-world

environment, there are more practical problems to be considered, including the challenge of unknown traffic discovery and model efficiency.

To be deployable to practical applications, an ETI model needs to discover unknown classes of traffic that were not anticipated in the training phase. However, most of the existing models are based on the closed-world assumption, which means that the training dataset is assumed to contain all the traffic classes in the model deployment environment. However, such assumption cannot be held in many practical applications. Consequently, the classifier trained with a closed set is easy to classify the samples from an unknown class to some class in the training set mistakenly. To solve this problem, researchers try to develop models supporting both known class sample classification and unknown class sample discovery.

Recently, a model named SEEN is proposed for unknown traffic discovery [2], which applies the Siamese network in the ETI area for the first time. SEEN can classify known traffic into the correct classes and distinguish unknown traffic. However, the traffic features that SEEN uses are relatively rough, and the network structure of SEEN is rather complicated, which limits its practical application.

Many ETI models with high classification accuracy use complex neural network structures, and some use models with a low degree of parallelism (e.g., RNN). These models have strong feature extraction ability, but they need a large amount of training data. Moreover, the efficiency of these models is not high enough, and the model update is complex, which makes it barely able to deal with the complex and changeable network environment.

In this paper, we focus on improving the real-time performance and flexibility of unknown traffic discovery. Inspired by SEEN and Transformer [3], we try to design appropriate inputs and neural networks for reducing the space and time complexity of our task. More precisely, the contribution of this paper includes the following:

- (1) We put forward a lightweight model LightSEEN for unknown traffic discovery. To the best of our knowledge, there are few lightweight deep learning methods in this area. The overall structure of LightSEEN is a Siamese network, and we use the multihead attention mechanism to capture the associations between packets and promote the degree of parallelism. Meanwhile, 1D-CNN is introduced for further feature extraction and integration, and we reuse part of the network structure to reduce parameter amount.
- (2) We design compact packet-level features as the network input, meaning that only the most informative field information and a small amount of payload are selected. In addition, to reduce the quality and length requirements of the packet stream, we try to shrink the number of packets used.
- (3) We analyse the efficiency and effectiveness of LightSEEN with abundant experiments on two public datasets. In the model, techniques including ResNet and layer normalization are used to increase the convergence speed of the model and avoid it from degradation. Experimental results show that the effectiveness of LightSEEN is overall at the same level as SEEN, whereas the parameter number of the former is 0.51% of the latter, and the average training time of the former is 37.9% of the latter.

The rest of this paper is organized as follows. In Section 2, we review the related work on unknown traffic discovery. In Section 3, we introduce the problem definition and the architecture of the Siamese network. The LightSEEN is presented in Section 4, followed by the corresponding analysis. In Section 5, we evaluate the efficiency and effectiveness of LightSEEN by conducting comparative experiments on two data sets. Finally, we conclude this paper in Section 6.

2. Related Work

In this section, under the background of encrypted traffic analysis, we briefly introduce the machine learning methods used to discover unknown traffic, which includes conventional machine learning methods and deep learning methods.

2.1. Conventional Machine Learning Methods. Firstly, we introduce the conventional machine learning methods for unknown traffic discovery briefly, mainly including semi-supervised and unsupervised methods.

Since under most circumstances, labeled samples are insufficient while unknown flows are sufficient, many existing results on unknown traffic identification use semi-supervised methods. In 2007, Erman et al. [4] firstly proposed a semisupervised classification method for traffic classification, in which the labeled training data was used to solve the problem of mapping from flow clusters to actual classes; thus, it could be used to classify known and unknown applications. In its subsequent work, Zhang et al. [5] proposed a robust statistical traffic classification (RTC) solution on the basis of [4] by combining supervised and unsupervised machine learning technology to solve the unknown Zero-Day application challenge in traffic classification. This method can identify the Zero-Day application traffic and accurately distinguish the applications of predefined classes, and its effectiveness was verified by comparative experiments. In the same year, Lin et al. [6] proposed UPCSS to detect unknown protocols, which was based on flow correlation and semisupervised clustering ensemble learning. Similarly, Ran et al. [7] proposed a semisupervised learning system for adaptive traffic classification in 2017, which adopted techniques including iterative semisupervised k -means and dynamically adding centers to select the optimal parameters and achieved high accuracy.

Considering that traffic data of known classes only accounts for a small part of the massive network traffic, researchers also try to extract unknown features from unlabeled data, namely, using unsupervised learning methods in network classification. Mapping the extracted clusters to classes is the main challenge in implementing these methods. In 2009, Este et al. [8] proposed a method based on SVM to solve multiclass classification problem, applied it to traffic classification, and carried out simple optimization, making the classifier trained with a small number of hundreds of samples classify traffic from different topological points on the Internet with high accuracy. Likewise, in 2018, Fu et al. [9] also proposed the FlowCop method based on multiple one-class classifiers, which could not only identify predefined traffic, but also detect undefined traffic with selected prominent features for each one-class classifier. Both of the solutions in [8, 9] are based on the method of multiple one-class classifiers, but the binary classifiers for each class in this method are heuristics. Moreover, this method relies on a predefined distance threshold, which may lead to unsatisfactory results. In 2019,

Le et al. [10] discussed the extent to which the self-organizing map (SOM) could be applied to network traffic analysis and malicious behaviors detection in practice. Experiment results showed that the approach could identify malicious behaviors both on network and service datasets used, and that it was also beneficial for security management with visualization capabilities for network/service data analytics.

The conventional machine learning methods have relatively low time and space cost, and they have scored some achievements in unknown traffic discovery. However, they suffer from a high dependence on expert experience for feature selection, which makes it laborious to build the models and limits their performance.

2.2. Deep Learning Methods. With the rapid development of deep learning and its wide application in various fields, many researchers have applied deep learning to unknown traffic identification.

In 2017, Ma et al. [11] used a CNN model to identify protocols in the complex network environment according to the protocol type of the application layer. Experiments showed that, in the payload information of about 200,000 traffic flows, the accuracy of identifying unknown protocol traffic was 86.05%. In 2019, Zhang et al. [12] proposed a method, DePCK, for identifying unknown traffic, which could divide the mixed unknown traffic into multiple clusters. Each cluster contained only one application traffic as much as possible, thus improving the clustering purity. This method uses a deep autoencoder to extract features from traffic and then lets flow correlation guide the process of pair-constrained k -means. In the same year, Zhu et al. [13] proposed a method using deep neural networks to select appropriate protocol flow statistical features with the help of known application layer protocols. They then used an improved semisupervised clustering algorithm to divide the protocols into different sets, achieving unknown protocol classification. In 2020, Wang et al. [14] proposed a CNN model for unknown protocol syntax analysis according to the characteristics of the bit-flow protocol data format. The model preprocesses the protocol data to obtain the image format data suitable for CNN and then lets CNN process the image data to obtain the prediction results of the unknown protocol. Besides, Zhang et al. [15] studied how extreme value theory (EVT) could be utilized in unknown network attack detection systems and brought out a network intrusion detection method. By fitting the activation of the known class to the Weibull distribution, the open-CNN model was constructed to estimate the pseudo-probability of the unknown class from the activation score of the known class to achieve the purpose of detecting unknown attacks. In addition, Yang et al. [16] proposed a transfer learning method using deep adaptation networks (DAN). This method first trains a CNN model on the unlabeled data set with sampling time-series features, then jointly trains the extended version of the model on the labeled and unlabeled samples, uses labeled samples of known traffic to improve the clustering purity of unknown

traffic. This method achieves a purity of 98.23% on two published data sets.

Most of the above works need prior knowledge of unlabeled traffic, leading to their insufficient capability of fine-grained identification of unknown traffic. To fix this problem, Chen et al. [2] firstly applied the Siamese network to unknown traffic discovery. Their method, SEEN, can classify known traffic into correct classes and distinguish unknown traffic. However, the rough traffic features that SEEN uses and its bloated network structure make it not suitable for a realistic environment. Compared with SEEN, the method in this paper uses simplified input and a carefully designed lightweight network, which makes it more practical.

3. Preliminaries

In this section, we briefly review the preliminaries used in our model, including the definition of unknown traffic discovery and the work process of the Siamese network.

3.1. Problem Definition. Encrypted traffic identification refers to using rules or models to give traffic samples correct labels. It can be conducted with multiple granularities, including packet, flow, and host level [17]. In this paper, we focus on bidirectional flow analysis. A bidirectional flow is composed of all packets with the same quintuple values, that is, source IP, source port, destination IP, destination port and transport layer protocol, in which the source and destination are interchangeable [18]. For a flow f_i containing N packets, it can be expressed as shown in equation (1). Typically, a model for traffic identification is trained with labeled flow data firstly, and the trained model is used to classify flow samples without label into classes correctly. In particular, the model may face the unknown traffic discovery problem in practice.

$$f_i = \{p_1^i, \dots, p_N^i\}. \quad (1)$$

Unknown traffic discovery requires a classifier to reject a flow from classes unseen during training rather than assigning it an incorrect label [19]. Given a training set $D_{\text{train}} = \{(f_1, y_1), \dots, (f_n, y_n)\}$, where f_i is the i th flow sample and $y_i \in \Omega^K = \{C_i^K, i = 1, \dots, P\}$ is its corresponding class label, the goal is to learn a classifier \mathcal{A} that can not only classify the samples from known classes correctly but also categorize samples from unknown classes as unknown. For a test sample f_* , whose actual class label is y_* , the ideal effect of \mathcal{A} is shown in the following equation:

$$\mathcal{A}(f_*) = \begin{cases} y_*, & \text{if } y_* \in \Omega^K, \\ \text{unknown}, & \text{if } y_* \notin \Omega^K. \end{cases} \quad (2)$$

3.2. Siamese Network. Siamese neural network is a class of network architectures that consists of two (or more) identical subnetworks. The subnetworks have the same structure with the same parameters and shared weights, which are synchronously updated. A loss function connect them at the end, which computes a similarity metric based on the Euclidean

distance between the feature representations produced by the subnetworks. A commonly used loss function in the Siamese network is the contrastive loss [20] defined as follows:

$$L(x_1, x_2, y) = \alpha(1 - y)D_w^2 + \beta y \max(0, m - D_w)^2, \quad (3)$$

where x_1 and x_2 are two samples, y is a binary label denoting whether the two samples are of the same class or not, α and β are constants, and m is the margin. $D_w = \|f(x_1; w_1) - f(x_2; w_2)\|_2$ is the Euclidean distance in the embedded feature space, f is an embedding function mapping a sample to the feature space via neural networks, and w_1 and w_2 are the learned networks weights.

Siamese network aims to let the loss function bring the output feature vectors of similar inputs closer and push those of dissimilar inputs away. Then, to decide if two inputs belong to the same class, one needs to determine a threshold value on the feature vector distance. If the distance between the two inputs is smaller than the threshold, they are treated as similar samples or from the same class. Otherwise, they are judged as from different classes.

4. The Lightweight Model for Unknown Traffic Discovery

In this section, we introduce the lightweight model for unknown traffic discovery we proposed, and Figure 1 displays the structure of the model. Besides, we illustrate the details of model training, model validation, model test, and system update and also analyse the space and time complexity of the model.

4.1. Model Structure. As mentioned in Section 3.2, the Siamese network is generally composed of two identical subnetworks, which are joined by the margin-based loss function at the end. Therefore, we only need to introduce the structure of one subnetwork to make the composition of the whole model clear.

In general, the subnetwork structure here consists of four parts, that is, preprocessing, feature embedding, attention module, and dense layer. Moreover, we will explain each part in detail.

Preprocessing: the purpose of preprocessing is to extract valuable packet information as features. For a raw PCAP file, the packets in it can be combined into flows according to the quintuple. Then, the flow can be preprocessed by extracting its packet features, which are carefully designed for the lightweight traffic analysis task.

Considering our lightweight detection task, we choose features as lean as possible. In detail, firstly, the three-way handshake is skipped since it can barely provide information for traffic classification. Besides, only the first $N = 3$ packets are picked to get features, which are most likely to disclose useful information. For each packet, $S = 5$ fields of features are concerned, namely, position, timestamp, direction, key flags in IP and TCP header, and packet payload, and the details and meanings of the features are as follows.

- (1) Position (one dimension): it is the sequence number of a packet in a flow, which provides order information.
- (2) Timestamp (one dimension): it marks the arrival time of a packet, which provides temporal information.
- (3) Direction (two dimensions): a bidirectional flow includes packets of two directions, namely, from source to destination and the reverse direction, which can be represented as [0,1] and [1,0].
- (4) Key flags in IP and TCP header (nine dimensions): the key flags include ip_len, ip_off, ip_ttl, PSH, URG, th_seq, th_urp, and th_win. The ip_len means packet length, the ip_off means fragment offset, the ip_ttl stands for Time to Live, the PSH indicates the data transmission pattern, the URG means urgent data, the th_seq means the relative sequence, the th_urp is the urgent data offset, and the th_win means the window size. Other flags are abandoned since they do not contribute to the task.
- (5) Packet payload (77 dimensions): if the payload is less than 77 bytes, it will be completed with zero bytes and conversely truncated to 77 bytes.

4.1.1. Feature Embedding. The embedding layer converts the raw packet features into packet vectors that can be better analysed by the neural networks. Since we have multiple features with different dimensions, how to integrate them is worth studying, and there are a wide range of choices. A recent work on this is [21], in which a method of unifying different kinds of features' dimensions was proposed. Besides, feature fusion can also be achieved by neural networks. However, in this work, to reduce model complexity and promote efficiency, we choose to concatenate the raw features directly as a simple embedding. Let $\{x_i, i = 1, \dots, S\}$ denote the raw features obtained from preprocessing and $\{\mathbf{p}_i, i = 1, \dots, N\}$ denote the packet vectors generated by feature embedding; then, we have

$$\mathbf{p}_i = \{x_1, \dots, x_S\}, \quad i = 1, \dots, M. \quad (4)$$

Then, the packet vector \mathbf{p}_i has a dimension of $d_p = 90$, and $\{\mathbf{p}_i, i = 1, \dots, N\}$ will be the input of the attention module, which has a dimension of $d = Nd_p = 270$.

4.1.2. Attention Module. The structures of the attention module and dense layer are shown in Figure 2. The design of the attention module is partly derived from the Transformer encoder, and we adjust the network to support lightweight unknown discovery. In brief, we leverage the multihead attention mechanism to capture the interactions between different packets, reuse the basic block, and introduce 1-dimensional CNN (1D-CNN) to accumulate information and decrease the scale of network parameters.

The attention module mainly consists of three components, namely, (1) multihead attention, (2) add & norm, and (3) 1D-CNN, which will be explained in detail.

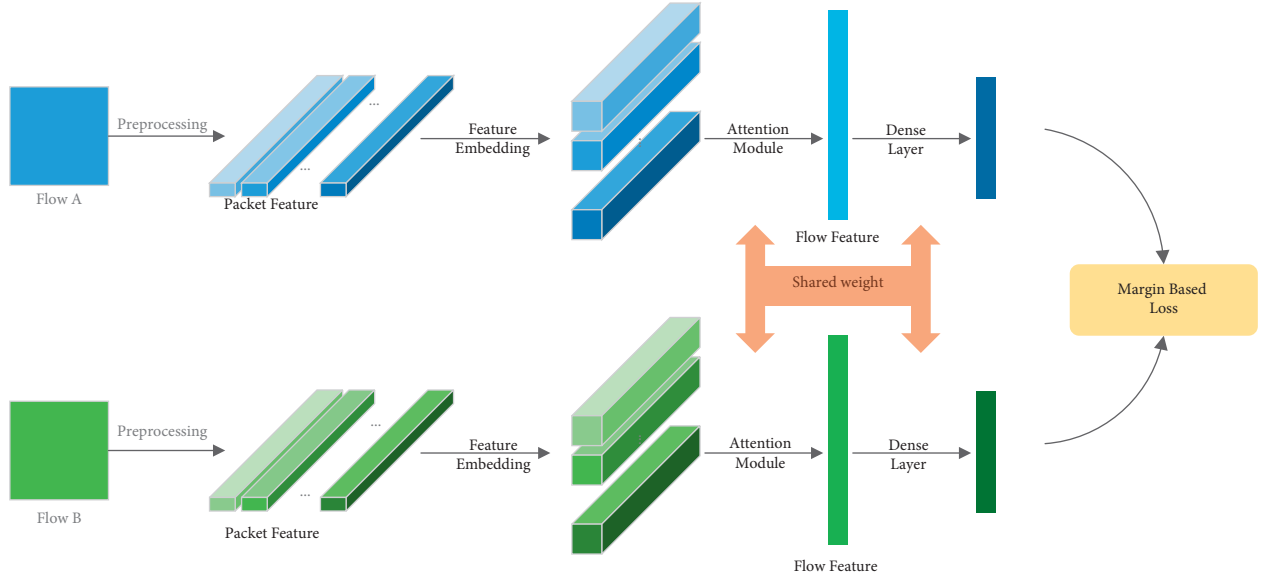


FIGURE 1: The mode structure of LightSEEN.

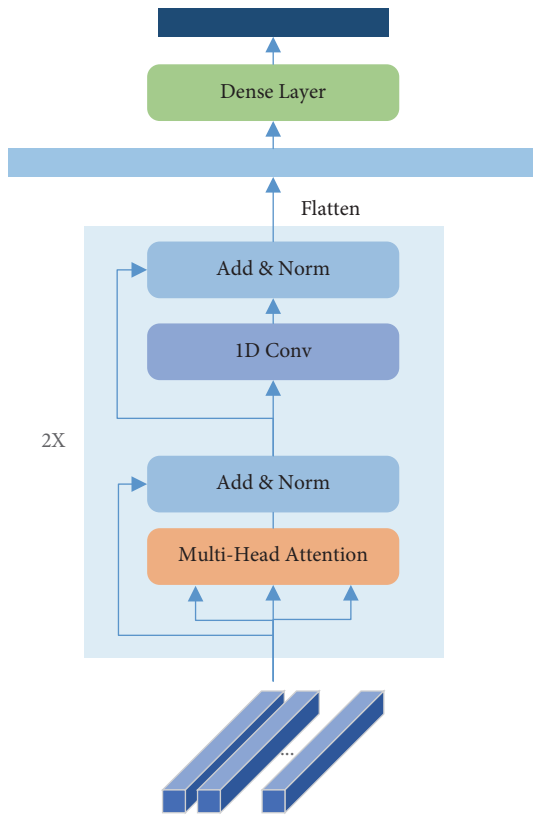


FIGURE 2: The construct of the attention module and the dense layer.

(1) *Multihead Attention*. The function of multihead attention is to jointly collect deep-level information of the input from multiple representation subspaces. We use packet k as an example to explain how the information delivered and organized efficiently to produce a new packet vector with deep-level features. The number of heads is denoted by H , so the

attention head $h \in \{1, \dots, H\}$. Let $\sigma^h(\cdot, \cdot)$ denote the relationship between two packets, and $\alpha_{k,l}^h$ denote the attention weight between packet k and l ; then,

$$\alpha_{k,l}^h = \frac{\sigma^h(\mathbf{p}_k, \mathbf{p}_l)}{\sum_{i=1}^N \sigma^h(\mathbf{p}_k, \mathbf{p}_i)}. \quad (5)$$

$\sigma^h(\cdot, \cdot)$ can be achieved by inner product or a neural network, and we choose inner product for better efficiency; hence,

$$\sigma^h(\mathbf{p}_k, \mathbf{p}_l) = \langle Q^h \mathbf{p}_k, K^h \mathbf{p}_l \rangle, \quad Q^h, K^h \in \mathbb{R}^{d' \times d}, \quad (6)$$

where Q^h and K^h are transformation matrices mapping the packet vector from original space \mathbb{R}^d into a new space $\mathbb{R}^{d'}$. Then, the representation of packet k of head h is

$$\hat{\mathbf{p}}_k^h = \sum_{i=1}^N \alpha_{k,i}^h (V^h \mathbf{p}_i), \quad V^h \in \mathbb{R}^{d' \times d}. \quad (7)$$

And, the packet vector in new space can be obtained by concatenating the $\hat{\mathbf{p}}_k^h$ of all heads:

$$\hat{\mathbf{p}}_k = \text{Concat}(\hat{\mathbf{p}}_k^1, \dots, \hat{\mathbf{p}}_k^H). \quad (8)$$

On the whole, the multihead attention mechanism updates the representation of all packets with the idea of weighted summation. For each packet, the weight is generated from its association with all the other packets in parallel, resulting in much fewer parameters and a much shorter running time. Besides, a packet is projected into different subspaces for capturing multiview feature associations. Since the computation of all the heads is also parallelized, it benefits for speeding up the model.

(2) *Add & Norm*. The add & norm part uses ResNet and layer normalization for avoiding network degradation and faster training. It has been discussed that the ResNet can make it

easier for information to flow between layers, including providing feature reuse in forward propagation and alleviating gradient signal disappearance in back propagation [22]. The effect of the ResNet on the representations of packet k can be expressed as

$$\bar{\mathbf{p}}_k = \text{ReLU}(\hat{\mathbf{p}}_k + \mathbf{W}_{\text{res}}\mathbf{p}_k), \quad (9)$$

where $\mathbf{W}_{\text{res}} \in \mathbb{R}^{d \times d}$ is a transformation matrix and ReLU is an activation function.

The subsequent layer normalization technique can normalize the distributions of mid-tier layers, making gradients smoother and generalization better.

(3) *1D-CNN*. The 1D-CNN part is designed for further mining the hidden patterns contained by the packet representation obtained from previous layers. Besides, compared with the fully connected network, CNN needs much fewer parameters. Specifically, the kernel size of the 1D-CNN layer is d and the channel number $r = d$, making the input and output dimensions consistent.

The output of 1D-CNN will be the input of another add & norm layer, and a basic block is composed of multihead attention, 1D-CNN and 2 add & norm layers, as is shown in Figure 2. The basic block is reused $T = 2$ times for a better balance between effectiveness and efficiency.

4.1.3. Dense Layer. Let $\{\tilde{\mathbf{p}}_i, i = 1, \dots, N\}$ denote the output of the whole attention module, which will be concatenated into a vector $\tilde{\mathbf{f}}$ as flow representation, shown in equation (10). Then, $\tilde{\mathbf{f}}$ will be fed into the dense layer, as equation (11) shows, and the output vector \mathbf{f} with length L will be the final flow vector.

$$\tilde{\mathbf{f}} = \text{Concat}(\tilde{\mathbf{p}}_1, \dots, \tilde{\mathbf{p}}_N), \quad (10)$$

$$\mathbf{f} = \text{ReLU}(\mathbf{W}_D \tilde{\mathbf{f}} + \mathbf{b}), \quad \mathbf{W}_D \in \mathbb{R}^{L \times N d}. \quad (11)$$

4.2. Model Training and Validation

4.2.1. Model Training. The model training means using labeled samples to train the Siamese network, and the first step is the pairwise dataset generation. Different from other networks, the input of the Siamese network is a pair of flows rather than a single data. Therefore, it is necessary to choose data from the labeled known class dataset to construct a new dataset containing positive and negative pairs. To be specific, a positive pair, which is labeled as 0, is a pair of flows that belong to the same class, and a negative pair with label 1 contains flows from different classes. To avoid the influence of imbalanced data, the ratio of positive to negative pairs is about 1 : 1. The model will learn a metric to tell similar and dissimilar pairs apart through these positive and negative samples.

Given a pair of flows f_i and f_j , the true label of the pair is denoted by l_t . Let x_i and x_j denote the corresponding raw flow features input to the network, and v_i and v_j denote the output of the network. The function of the network can be

represented as $F_L(x; \theta)$, where θ denotes the parameters; then, we have $v_i = F_L(x_i; \theta)$ and $v_j = F_L(x_j; \theta)$. The distance between f_i and f_j , denoted by $D_{i,j}$, can be calculated as follows:

$$D_{i,j} = D(F_L(x_i; \theta), F_L(x_j; \theta)) = \left[\sum_z (v_{iz} - v_{jz})^2 \right]^{1/2}. \quad (12)$$

With the pairwise dataset generated and the hyperparameter margin m set, the model can be trained for the binary classification problem. The margin-based loss function is shown in Section 3.2, which encourages positive pairs to be close together in the space of network mapping while pushing negative pairs apart. Let $\alpha = \beta = 1/2$; the loss function for our model training is shown as follows:

$$L(x_i, x_j, l_t) = \frac{1}{2} (1 - l_t) D_{i,j}^2 + \frac{1}{2} l_t \max(0, m - D_{i,j})^2. \quad (13)$$

4.2.2. Model Validation. The model validation means validating that the model can differentiate the positive and negative pairs with high accuracy. The positive and negative pairs are also generated from known classes. However, it is suggested that the pairs for validation should be avoided from overlapping with those in the training process. An appropriate method is generating a group of nonredundant pairs from the known classes and splitting the group into training and validation datasets. Besides, a threshold t should be determined through experience or attempts. Let l_t denote the true label of a pair and l_p denote the predicted label. If the Euclidean distance $D_{i,j}$ of a flow pair (f_i, f_j) is beyond the threshold, then the predicted label is $l_p = 1$; namely, the pair is judged as negative. Otherwise, the pair is judged as positive with the predicted label $l_p = 0$, as is shown in equation (14). Then, we can compare l_t and l_p of each pair, if they are the same, the judgment of the model is correct. The model validation can be used to validate the training result of the model and adjust the value of t and even the margin.

$$l_p = \begin{cases} 1, & \text{if } D_{i,j} > t, \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

4.3. Model Test and System Update

4.3.1. Model Test. Model test means using the trained LightSEEN model for traffic classification and unknown discovery; that is, it should not only classify the flows from known classes correctly but also detect flows from unknown classes. For a flow f , the actual label of f is denoted by $\Phi(f)$, and its predicted label is denoted by $\varphi(f)$. Recall that Ω^K is the set of known classes labels; then, the known flow dataset can be expressed by O_f^K in equation (15). The defined distance between flows is not enough for the task, a distance between a test flow sample f^* and a known class C_i^K must be defined. We use the same distance as [2] uses, which is

defined as the average distance between the test sample f^* and q samples $\{f_{ij}, j = 1, \dots, q\}$ randomly chosen from the class C_i^K . The calculation of the sample-class distance is expressed by equation (16). Then, there is a known class C_*^K which is the closest to f^* ; let D_{f^*} denote the distance. If D_{f^*} is not larger than the preset threshold t , it is decided that f^* belongs to C_*^K . Otherwise, f^* belongs to no known class; namely, its class is unknown. The model test algorithm is shown in Algorithm 1.

$$O_f^K = \{f | \Phi(f) \in \Omega^K = \{C_1^K, \dots, C_P^K\}\}, \quad (15)$$

$$D(f^*, C_i^K) = \frac{1}{q} \sum_{j=1}^q D(f^*, f_j), \Phi(f_j) = C_i^K, \quad j = 1, \dots, q. \quad (16)$$

The model test dataset is made up equally of flow samples from known and unknown classes, and the related metrics will be introduced in Section 5.1.

4.3.2. System Update. To update the system, an unsupervised framework should be leveraged to divide the detected unknown traffic into multiple clusters, which can be used as a supplement to known classes. The trained network can be used as an encoder to convert the original flow data to high-level feature vectors, which can be clustered by existing algorithms like k -means. After that, the clusters are identified through manually labeling and used to complement the system's identification area. For instance, if we want to add a new class C^* to the known classes, we only need to use samples from C^* and known classes to generate positive and negative pairs, with at least one sample from C^* in each pair. Then, we use the generated pairs to retrain the model, and we get a model for $P + 1$ known classes.

4.4. Space and Time Complexity Analysis. For deep learning models, the space complexity is related to its parameter amount, and the time complexity depends on its inner structure. Table 1 shows the space and time complexity of LightSEEN, and the corresponding analysis is as follows.

4.4.1. Space Complexity. In our method, there is no parameter in the embedding layer. For the attention module, the parameters are mainly in the weight matrices, and the scale is about $T \times \{Q^h, K^h, V^h, \mathbf{W}_{res}, \mathbf{W}_{1D-CNN}\}$, which is $O(\text{THdd}' + \text{Tdr})$. The parameter scale of the dense layer is $O(\text{NdL})$. Therefore, the total space complexity is $O(\text{THdd}' + \text{Tdr} + \text{NdL})$.

4.4.2. Time Complexity. In the attention module, for each head, the time cost for attention weight calculation is $O(\text{Ndd}' + N^2 d')$ and that for combinatorial feature formulation is $O(N^2 d d')$. Besides, the time complexity of the add & norm, 1D-CNN, and dense layer are $O(\text{Hdd}')$, $O(\text{Nd}^2 r)$, and $O(\text{NdL})$, respectively. Considering that the

attention module reuse T times, the total space complexity is $O(\text{Tdd}'(H + N^2) + \text{TNd}^2 r + \text{NdL})$.

5. Experimental Evaluation

In this section, we evaluate the effectiveness and efficiency of LightSEEN. Since LightSEEN is built to enhance the real-time performance and flexibility of deep learning based unknown traffic discovery, we mainly compare the LightSEEN method with SEEN [2].

5.1. Experiment Setup

5.1.1. Datasets and Partition Strategy. We tested the performance of LightSEEN on two extensively used public traffic datasets, namely, USTC-TFC2016 [23] and ISCXVPN2016 [24]. As is shown in Table 2, USTC-TFC2016 contains 20 classes of traffic, of which half are malware traffic classes. The ISCXVPN2016 dataset includes seven classes of regular encrypted traffic and seven classes of traffic through the VPN encrypted tunnel, and we use 12 of them to conduct experiments.

The partition strategy for known and unknown sets is the same as that in [2], namely, some classes are manually set as unknown classes, including three malware classes and three normal classes traffic from USTC-TFC2016, two VPN classes, and two non-VPN classes from ISCXVPN2016.

Experiment Environment and Details: as for the experiment environment, we used PyTorch 1.8 to implement the structure of LightSEEN. Note that the training and testing processes were performed on a Linux machine (Ubuntu 16.04 LTS) with 32 GB RAM and GeForce Gtx1080. The training process is guided by minimizing the contrastive loss, and we take the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The parameters of LightSEEN are shown in Table 3. The dropout strategy is applied with a keep proportion of 0.9 for the multihead attention part and 0.6 for the dense layer. The learning rate is 0.0002, and batch size in model training is 128. For the balance of efficiency and effectiveness, we set the margin $m = 6$ for USTC-TFC2016 and $m = 12$ for ISCXVPN2016.

5.1.2. Evaluation Metrics. The performance of LightSEEN mainly includes the efficiency and effectiveness of unknown discovery. To evaluate its efficiency, we count the training and test time per 100 batches, and the average training and test time are used as evaluation metrics. As to its effectiveness, four evaluation metrics are used [25]: purity rate (PR), accuracy (Acc), false detection rate (FDR), and true detection rate (TDR). To illustrate the metrics, some other symbols are defined. KP (known positive) denotes the number of the known class flows correctly identified, KN (known negative) denotes the number of the known class flows mistaken for other known classes, UP (unknown positive) denotes the number of unknown class flows detected, and UN (unknown negative) denotes the number of unknown flows wrongly classified as known. Then, the metrics can be computed with these statistics as follows.

Input: test flow sample f^* , known class flow dataset O_j^K , number of samples for average distance calculation q , threshold t .
 (1) Calculate the distance between f^* and each class $\{D(f^*, C_i^K), i = 1, \dots, P\}$;
 (2) Find the class with the shortest distance away from f^* , denoted as C_*^K ;
 (3) If $D(f^*, C_*^K) \leq t$, $\varphi(f^*) = C_*^K$. Otherwise, $\varphi(f^*) = \text{unknown}$.
 Output: the predicted class $\varphi(f^*)$.

ALGORITHM 1: Model test algorithm.

TABLE 1: Space and time complexity of the proposed model.

Layer	Space complexity	Time complexity
Attention module	$O(\text{THdd}' + \text{Tdr})$	$O(T(N^2 + H)dd' + \text{TNd}^2r)$
Dense layer	$O(\text{NdL})$	$O(\text{NdL})$
All	$O(\text{THdd}' + \text{Tdr} + \text{NdL})$	$O(\text{Tdd}'(N^2 + H) + \text{TNd}^2r + \text{NdL})$

TABLE 2: USTC-TFC2016 dataset and ISCXVPN2016 dataset.

Dataset	Labels
USTC-TFC2016	Cridex, Ceodo, Hitbot, Miuref, Neris, Nsis-ay, Shifu, Tinba, Virut, Zeus, BitTorrent, Facetime, FTP, Gmail, MySQL, Outlook, Skype, SMB, Weibo, and WorldOfWarcraft
ISCXVPN2016	Chat, Email File, P2P, Streaming, VoIP, VPN-Chat, VPN-Email, VPN-File, VPN-P2P, VPN-Streaming, and VPN-VoIP

TABLE 3: The parameters of LightSEEN.

Meaning	Parameter	Value
Number of packets	N	3
Times of attention module reuse	T	2
Number of headers	H	3
Dimension of embedding	d/r	270
Dimension of attention head mapping	d_t	90
Dimension of flow vectors	L	200

From the equations, it is easy to see that the solution with high PR, Acc, and TDR and low FDR has favorable performance.

$$\begin{aligned}
 \text{PR} &= \frac{\text{KP} + \text{UP}}{\text{KP} + \text{KN} + \text{KU} + \text{UP} + \text{UN}}, \\
 \text{Acc} &= \frac{\text{KP}}{\text{KP} + \text{KN} + \text{KU}}, \\
 \text{FDR} &= \frac{\text{KU}}{\text{KP} + \text{KN} + \text{KU}}, \\
 \text{TDR} &= \frac{\text{UP}}{\text{UP} + \text{UN}}.
 \end{aligned} \tag{17}$$

Besides, the clustering purity (CP) is used to evaluate the performance of LightSEEN as a feature extractor, which will be explained in detail in Section 5.5. The definition of Clustering Purity is shown in equation (18), where $|D|$ is the number of samples, $\Omega = \{w_i, i = 1, \dots, K\}$ is the set of clusters, and $C = \{c_j, j = 1, \dots, J\}$ is the set of classes.

$$\text{CP}(\Omega, C) = \frac{1}{|D|} \sum_{i=1, \dots, K} \max_{j=1, \dots, J} |w_i \cap c_j|. \tag{18}$$

5.2. *Selection of Hyperparameters.* There are two hyperparameters that are not used in model training but indispensable for unknown traffic discovery (i.e., k (the number of compared samples from each class to calculate class average distance) and t (the threshold for determining whether the test sample belongs to a certain class)). To obtain the reasonable range of t , we use the trained model to predict pairwise Euclidean distances of the two datasets and display the corresponding histograms in the range $[0, 7]$ in Figure 3. We set green bars for positive pairs and orange bars for negative pairs. It can be seen that the distances of positive pairs are close to 0, and those of negative pairs are mostly far from 0. The coincided field of green and orange bars mainly lies in $[0, 1.5]$ for USTC-TFC2016 and $[0.5, 2.5]$ for ISCXVPN2016; thus, we choose $t = 1.2$ for USTC-TFC2016 and $t = 2.1$ for ISCXVPN2016. As to the number of samples k , experiments show that its influence on the performance of LightSEEN is small. Therefore, we adopt the same setting as [2] for the convenience of comparison between LightSEEN and SEEN, meaning that we set $k = 10$ for both datasets.

5.3. *Effectiveness Analysis.* To observe the effectiveness of LightSEEN under different situations, we change the percentage of unknown classes in the model test procedure from 10% to 50% and compare different models' performance. Figures 4 and 5 show the result comparison among LightSEEN, SEEN, and a one-class SVM method [8] on the USTC-TFC2016 and ISCXVPN2016 datasets, respectively. We set the green bars for LightSEEN, orange bars for SEEN, and blue bars for one-class SVM. It can be seen that the comparative advantages among the three methods are similar on the two datasets.

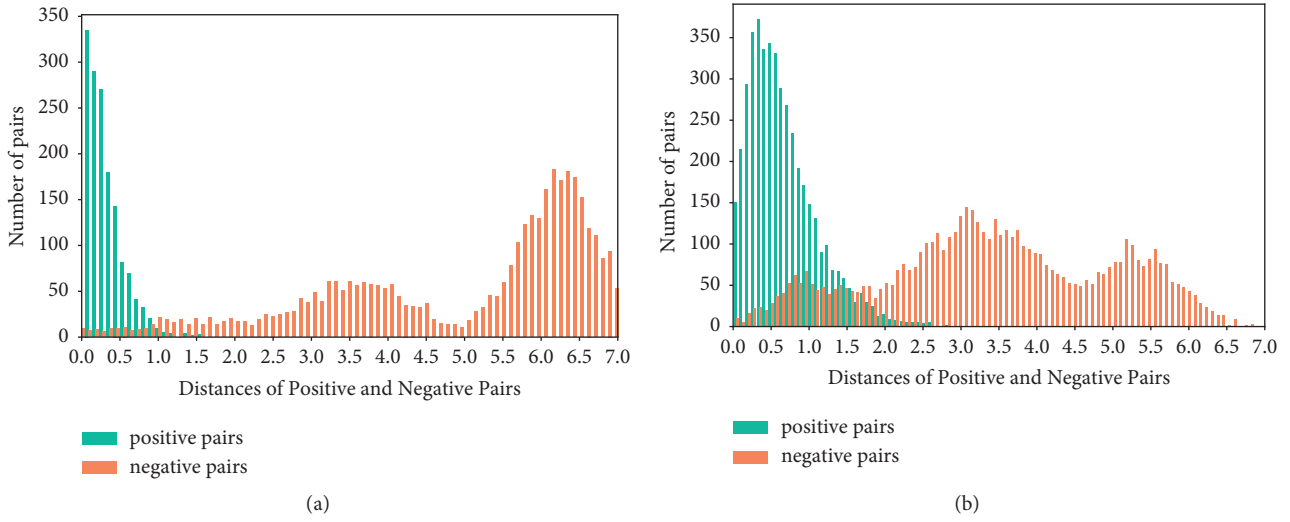


FIGURE 3: Part of the distance histogram of positive and negative pairs from two datasets. (a) USTC-TFC2016. (b) ISCXVPN2016.

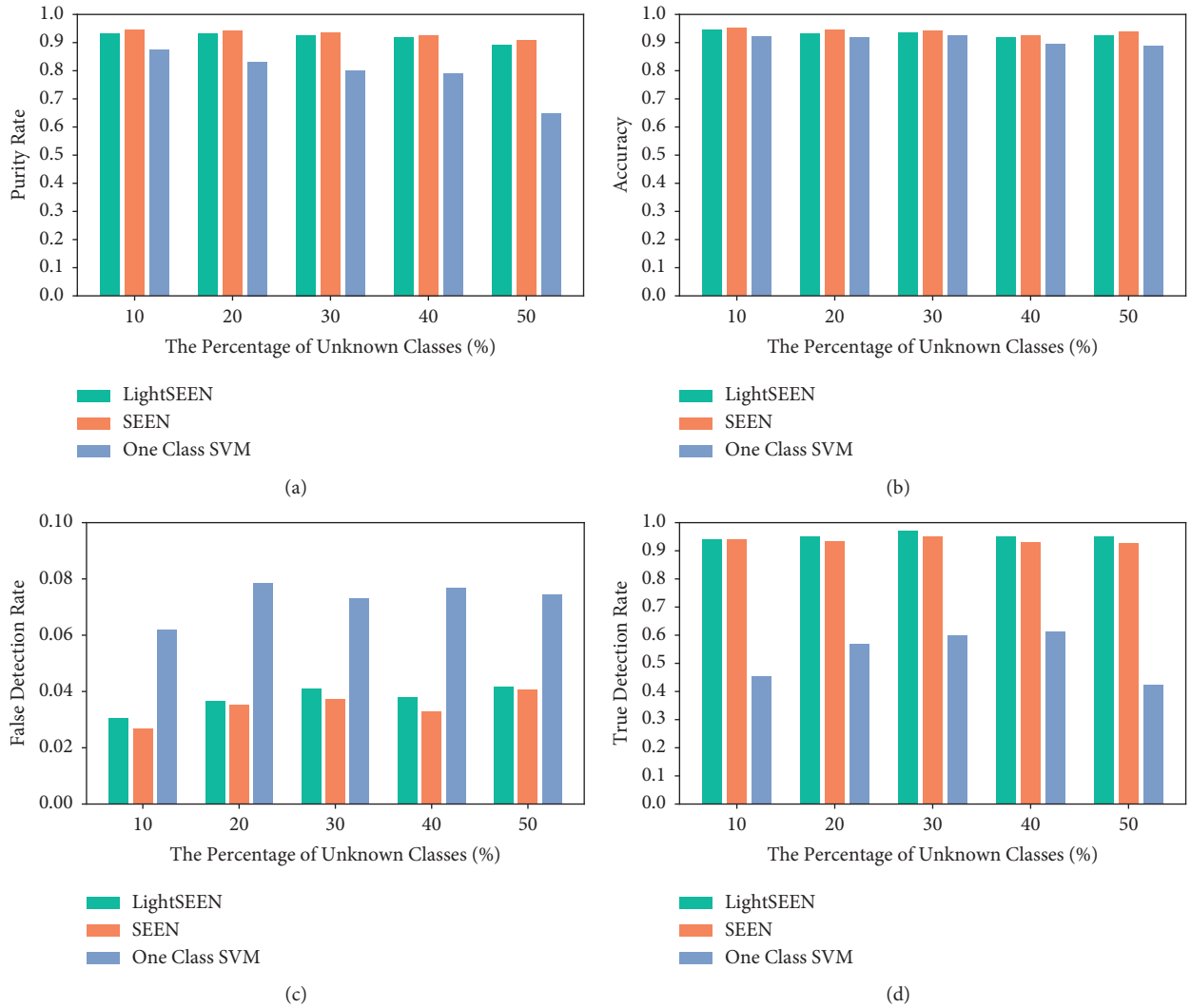


FIGURE 4: Performance comparison of different methods for the USTC-TFC2016 dataset.

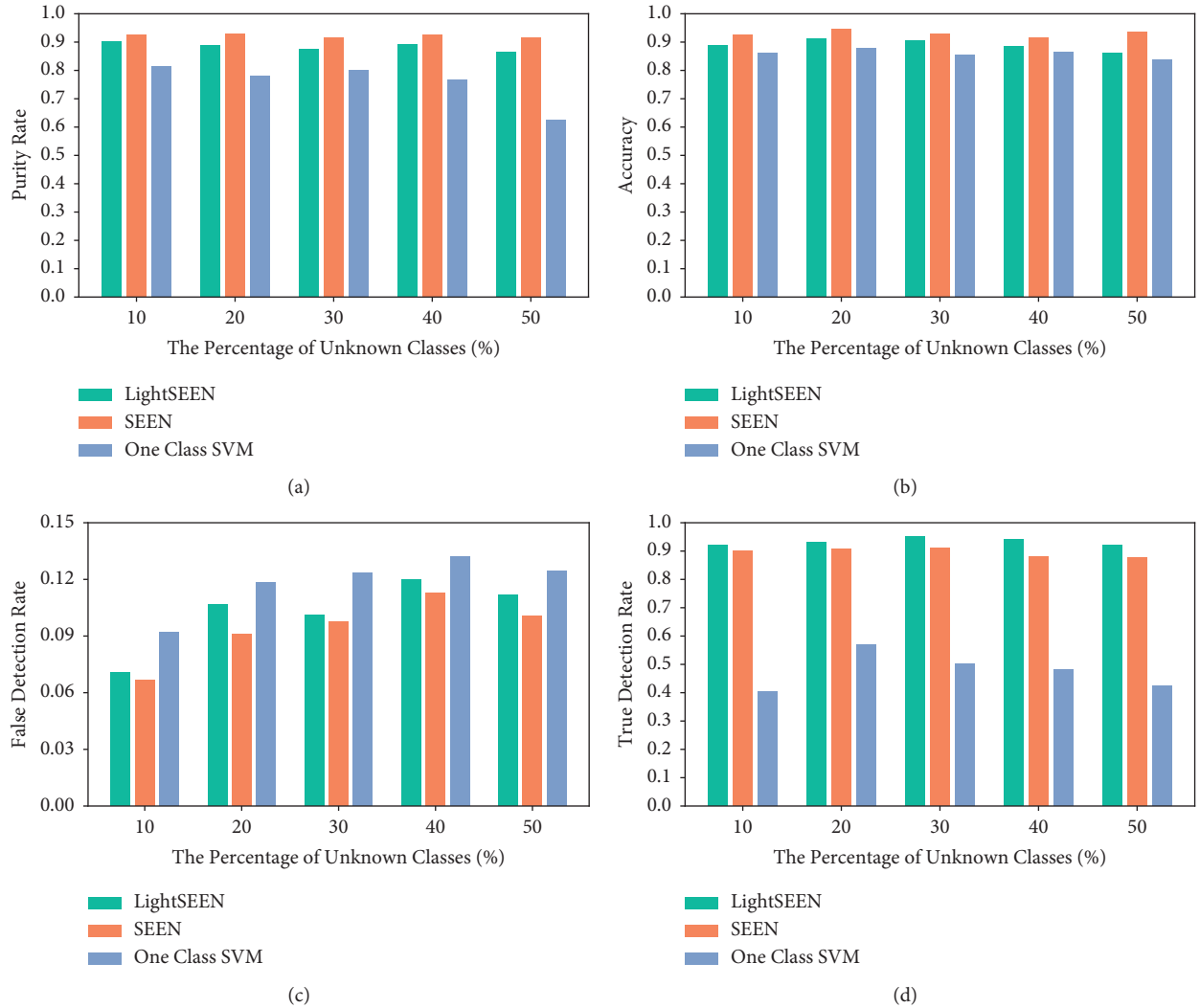


FIGURE 5: Performance comparison of different methods for the ISCXVPN2016 dataset.

To be specific, firstly, both of the purity rates (PR) of LightSEEN and SEEN are higher than that of one-class SVM and mostly above 0.9. Although the PR of LightSEEN is slightly lower than that of SEEN, they are very close, and both of them are stable with the percentage of unknown increases. The situation of the accuracy (Acc) result is almost the same as that of the PR result; that is, the result of SEEN is slightly better than LightSEEN on Acc, and both of them outperform the one-class SVM method. As for the false detection rate (FDR), SEEN has the lowest bar, LightSEEN's bar is slightly higher, and the one-class method's bar is the highest. Since a lower FDR means better performance, still SEEN is the best. However, for the true detection rate (TDR), LightSEEN is higher than SEEN and the one-class method. Note that in some reality applications like intrusion detection, the TDR is significantly crucial, meaning that LightSEEN is the best choice under these circumstances.

In summary, the effectiveness of LightSEEN on evaluation metrics is overall at the same level as SEEN and sometimes even better, meaning that its effectiveness is validated.

5.4. Efficiency Analysis. We demonstrate the efficiency of LightSEEN from three aspects, namely, quantity of parameters, average training time, and average test time. To promote the model efficiency, we take measures including multihead attention and reuse of the basic block in the attention module. Table 4 shows the comparison results of efficiency between LightSEEN and SEEN. The parameter number of our LightSEEN model is about 648000, which is 0.51% of that of SEEN. Besides, LightSEEN's average training time is 37.4 ms, which is 37.9% of that of SEEN. And its average test time is also obviously shorter. Through the efficiency analysis results, we can draw the conclusion that we substantially reduce the scale of model parameters and training time cost in LightSEEN, whose efficiency has been validated.

5.5. Unknown Clustering. In this part, we explore the performance of LightSEEN as a feature extractor. After detecting unknown traffic, we can separate them into different groups through clustering algorithm and update the

TABLE 4: The comparison results of efficiency.

Index	LightSEEN	SEEN
Quantity of parameters	648 k	126180 k
Average training time (ms)	37.4	98.6
Average test time (ms)	10.5	23.8

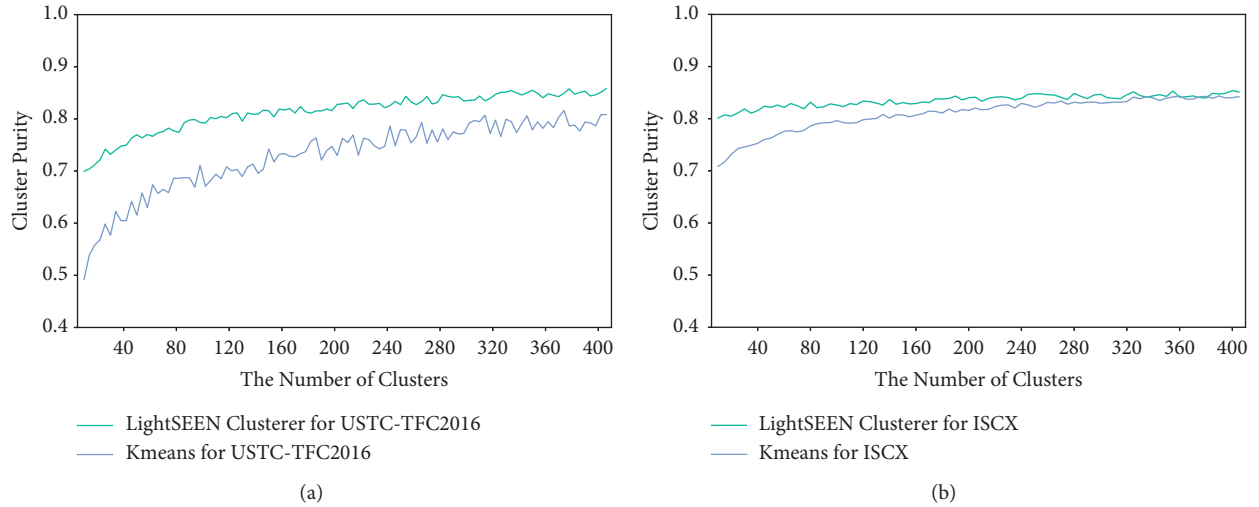


FIGURE 6: Clustering purity comparison of different methods for two datasets. (a) USTC-TFC2016. (b) ISCXVPN2016.

model as mentioned in Section 4.3. Moreover, we suggest that the clustering algorithm should be operated on the flow vectors output by the network of LightSEEN rather than the raw flow features.

We compare the clustering purity of traffic data with and without the processing of the network in LightSEEN, and Figure 6 shows the corresponding result. The blue line reveals the result of directly applying k -means to the flow vector composed by concatenating raw packet feature vectors and the green line for operating on the output of the trained network instead. It indicates that LightSEEN can extract deep features that more discriminative than raw features.

6. Conclusion

In this paper, we propose a lightweight model for unknown traffic discovery. Specifically, the model takes the cautiously selected packet features as input, adopts the Siamese network architecture, and guides the training process by contrastive loss. To capture the associations between packets and improve the parallel degree of the model, we use the multihead attention mechanism within the network. Besides, we introduce 1D-CNN, ResNet, and layer normalization, and reuse the basic modules to facilitate the model convergence with a limited number of parameters. The experimental results show that the model is effective and efficient.

In the future, further work can be done on open-set traffic recognition. Firstly, the contrastive loss in our model can be replaced by better loss functions (e.g., circle loss [26] and ArcFace loss [27]). Furthermore, the model can be applied to other practical tasks, including intrusion

detection and malicious traffic discovery. When intruders and attackers carry out actions against information systems, there will be anomalous traffic, which can be seen as unknown traffic and detected by unknown discovery systems.

Data Availability

The USTC-TFC2016 and ISCXVPN2016 data used to support the findings of this study are included within the article.

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 National Natural Science Foundation of China (nos. 61772548 and 61862011) and in part by the Guangxi Science and Technology Foundation (nos. 2018GXNSFAA138116 and 2019GXNSFGA245004).

References

- [1] M. Roughan, S. Sen, O. Spatscheck, and N. G. Duffield, "Class-of-service mapping for qos: a statistical signature-based approach to IP traffic classification," in *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference, IMC 2004*, A. Lombardo and J. F. Kurose, Eds., ACM, Taormina Sicily, Italy, pp. 135–148, October 2004.
- [2] Y. Chen, Z. Li, J. Shi, G. Gou, C. Liu, and G. Xiong, "Not afraid of the unseen: a siamese network based scheme for unknown traffic discovery," in *Proceedings of the IEEE Symposium on*

- Computers and Communications, ISCC 2020*, pp. 1–7, IEEE, Rennes, France, July, 2020.
- [3] A. Vaswani, N. Shazeer, N. Parmar et al., “Attention is all you need,” in *Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, L. Beach, U. S. A. CA, I. Guyon et al., Eds., pp. 5998–6008, Long Beach, CA, USA, December, 2017.
 - [4] J. Erman, A. Mahanti, M. F. Arlitt, I. Cohen, and C. L. Williamson, “Offline/realtime traffic classification using semi-supervised learning,” *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, 2007.
 - [5] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, “Robust network traffic classification,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1257–1270, 2015.
 - [6] R. Lin, O. Li, Q. Li, and Y. Liu, “Unknown network protocol classification method based on semi-supervised learning,” in *Proceedings of the 2015 IEEE International Conference on Computer and Communications (ICCC)*, pp. 300–308, IEEE, Chengdu, China, October 2015.
 - [7] J. Ran, X. Kong, G. Lin, D. Yuan, and H. Hu, “A self-adaptive network traffic classification system with unknown flow detection,” in *Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1215–1220, IEEE, Chengdu, China, December 2017.
 - [8] A. Este, F. Gringoli, and L. Salgarelli, “Support vector machines for TCP traffic classification,” *Computer Networks*, vol. 53, no. 14, pp. 2476–2490, 2009.
 - [9] N. Fu, Y. Xu, J. Zhang, R. Wang, and J. Xu, “Flowcop: detecting “stranger” in network traffic classification,” in *Proceedings of the 27th International Conference on Computer Communication and Networks, ICCCN*, pp. 1–9, IEEE, Hangzhou, China, August, 2018.
 - [10] D. C. Le, N. Zincir-Heywood, and M. I. Heywood, “Unsupervised monitoring of network and service behaviour using self organizing maps,” *Journal of Cyber Security and Mobility*, pp. 15–52, 2019.
 - [11] R. Ma and S. Qin, “Identification of unknown protocol traffic based on deep learning,” in *Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1195–1198, IEEE, Chengdu, China, December 2017.
 - [12] Y. Zhang, S. Zhao, and Y. Sang, “Towards unknown traffic identification using deep auto-encoder and constrained clustering,” in *Proceedings of the 2019 19th International Conference Computational Science - ICCS*, J. M. F. Rodrigues, P. J. S. Cardoso, J. M. Monteiro et al., Eds., vol. 11536, pp. 309–322, Springer, Faro, Portugal, June, 2019.
 - [13] P. Zhu, S. Zhang, H. Luo, and Z. Wu, “A semi-supervised method for classifying unknown protocols,” in *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1246–1250, IEEE, Chengdu, China, March 2019.
 - [14] Y. Wang, B. Bai, X. Hei, L. Zhu, and W. Ji, “An unknown protocol syntax analysis method based on convolutional neural network,” *Transactions on Emerging Telecommunications Technologies*, Wiley, Hoboken, NJ, USA, 2020.
 - [15] Y. Zhang, J. Niu, D. Guo, Y. Teng, and X. Bao, “Unknown network attack detection based on open set recognition,” *Procedia Computer Science*, vol. 174, pp. 387–392, 2020.
 - [16] Z. Yang and W. Lin, “Unknown traffic identification based on deep adaptation networks,” in *Proceedings of the 45th IEEE LCN Symposium on Emerging Topics in Networking, LCN Symposium 2020*, H. Tan, L. Khoukhi, and S. Oteafy, Eds., pp. 10–18, IEEE, Sydney, Australia, November 2020.
 - [17] S. Rezaei and X. Liu, “Deep learning for encrypted traffic classification: an overview,” *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.
 - [18] A. Dainotti, A. Pescapè, and K. C. Claffy, “Issues and future directions in traffic classification,” *IEEE Netw*, vol. 26, no. 1, pp. 35–40, 2012.
 - [19] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, “Toward open set recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013.
 - [20] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, pp. 539–546, IEEE Computer Society, San Diego, CA, USA, June 2005.
 - [21] W. Song, C. Shi, Z. Xiao et al., W. Zhu, “Automatic feature interaction learning via self-attentive neural networks,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019*, D. Tao, X. Cheng, P. Cui et al., Eds., ACM, Beijing, China, pp. 1161–1170, November 2019.
 - [22] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *Proceedings of the Computer Vision - ECCV 2016 - 14th European Conference*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908, pp. 630–645, Springer, Amsterdam, The Netherlands, October, 2016.
 - [23] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning,” in *Proceedings of the 2017 International Conference on Information Networking, ICOIN 2017*, pp. 712–717, IEEE, Da Nang, Vietnam, January, 2017.
 - [24] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of encrypted and VPN traffic using time-related features,” in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016*, O. Camp, S. Furnell, and P. Mori, Eds., SciTePress, Rome, Italy, pp. 407–414, February, 2016.
 - [25] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, “An effective network traffic classification method with unknown flow detection,” *IEEE Transaction Network Service Management*, vol. 10, no. 2, pp. 133–147, 2013.
 - [26] Y. Sun, C. Cheng, Y. Zhang et al., “Circle loss: a unified perspective of pair similarity optimization,” in *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*, pp. 6397–6406, IEEE, Seattle, WA, USA, June, 2020.
 - [27] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “Arcface: Additive angular margin loss for deep face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pp. 4690–4699, Computer Vision Foundation/IEEE, Long Beach, CA, USA, June, 2019.

Research Article

Website Fingerprinting Attacks Based on Homology Analysis

Maohua Guo  and Jinlong Fei 

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China

Correspondence should be addressed to Jinlong Fei; feijinlong_2021@163.com

Received 11 June 2021; Revised 31 August 2021; Accepted 24 September 2021; Published 4 October 2021

Academic Editor: Weiwei Liu

Copyright © 2021 Maohua Guo and Jinlong Fei. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Website fingerprinting attacks allow attackers to determine the websites that users are linked to, by examining the encrypted traffic between the users and the anonymous network portals. Recent research demonstrated the feasibility of website fingerprinting attacks on Tor anonymous networks with only a few samples. Thus, this paper proposes a novel small-sample website fingerprinting attack method for SSH and Shadowsocks single-agent anonymity network systems, which focuses on analyzing homology relationships between website fingerprinting. Based on the latter, we design a Convolutional Neural Network-Bidirectional Long Short-Term Memory (CNN-BiLSTM) attack classification model that achieves 94.8% and 98.1% accuracy in classifying SSH and Shadowsocks anonymous encrypted traffic, respectively, when only 20 samples per site are available. We also highlight that the CNN-BiLSTM model has significantly better migration capabilities than traditional methods, achieving over 90% accuracy when applied on a new set of monitored sites with only five samples per site. Overall, our experiments demonstrate that CNN-BiLSTM is an efficient, flexible, and robust model for website fingerprinting attack classification.

1. Introduction

With the continuous development of Internet technologies, privacy protection has become one of the most critical concerns. Thus, a continuously increasing number of users protect their anonymity while browsing the Internet by utilizing anonymous network communication systems. However, current research [1–10] shows that privacy can be compromised even though clients use privacy-enhancing technologies such as Shadowsocks [11], I2P [12], Tor [13], Anonymizer [14], SSH, and VPN. Among several cyberattacks compromising anonymity, the website fingerprinting attack is one of the most representative ones. The core idea of this type of attack is that although the user's communication content is encrypted when visiting different websites, the traffic characteristics generated by each website are unique due to each web page content, e.g., web code, images, scripts, and style sheets. Therefore, the attacker can analyze the anonymous traffic and infer the user's network behavior by passively extracting the traffic between the user and the anonymous network portal using the WF attack.

Current literature [1–5, 7, 8, 10, 15–17] considers website fingerprinting attacks a classification problem. Indeed, the attacker first builds a unique fingerprint model for each website and trains a suitable classifier using the fingerprint features, which can then be used to classify the collected user traffic. Early researchers used machine learning models such as Support Vector Machines [16] (SVM), k-Nearest Neighbors (k-NN) [10], and Random Forests [8], managing an attack accuracy of up to 90%. Nevertheless, in these techniques, the model performance mainly depends on handcrafted features. With the wide application of deep learning techniques in the field of traffic identification, attackers have applied deep learning models to website fingerprinting attacks [1–5, 7, 9], dramatically increasing the attack accuracy and effectively solving the challenging problem of feature extraction and selection. Although the advent of deep learning models has improved the attack accuracy, researchers need to collect hundreds of training samples for each website to enable the neural network to extract high-dimensional fingerprint features. Involving a large training dataset is crucial because when the training sample size is small, the model suffers significantly from

overfitting affecting the model's training process. Simultaneously, the traditional deep models are less flexible, with their performance dropping dramatically when applied to an entirely new classification task.

Spurred by the drawbacks of current deep learning methods, we propose a homology analysis-based approach for website fingerprinting attacks that employ a Siamese Networks [18] structure. Our deep learning architecture analyzes the homology relationship between website fingerprinting features and significantly reduces the training samples required for model training and managing an improved migration capability for the model. The main contributions of our work are as follows:

- (i) We study and propose a homology analysis-based website fingerprinting attack model, relying on a Convolutional Neural Network-Bidirectional Long Short-Term Memory (CNN-BiLSTM), which achieves 94.8% and 98.1% attack accuracy in a closed world composed of encrypted traffic from SSH and Shadowsocks anonymity networks, respectively, with only 20 training samples per site. The performance of the proposed architecture is significantly better compared to traditional methods.
- (ii) We innovatively construct one-hot matrices by sequence symbolization to represent the direction, size, and time interval attributes exposed in the traffic sequences. This strategy improves the data feature dimensionality and the fault tolerance for sample burst features.
- (iii) Compared to previous studies, we design a more challenging scenario to evaluate the model's migration capability. Specifically, we complete training using SSH anonymous network encrypted traffic and then utilize the trained model to classify Shadowsocks anonymous network encrypted traffic. The results demonstrate that, with only five sample attacks per site, our technique exceeds 90% classification accuracy.

The remainder of this paper is organized as follows. Section 2 summarizes and reviews previous approaches to website fingerprinting attacks. In Section 3, we present the threat model for website fingerprinting attacks and the design of the CNN-BiLSTM model. Section 4 summarizes the datasets used and the data processing methods, while Section 5 provides the results of our experiments and the corresponding analysis. The limitations of our work and directions for future research are discussed in Section 6. Section 7 concludes this work.

2. Background and Related Work

Website fingerprinting attacks use a passive traffic analysis technique. The attacker first configures a network environment similar to the monitored target, exploits the same anonymous network encryption proxy to access each site in the monitored set, and collects adequate training samples. After that, the attacker builds a fingerprint library for each

monitored site and identifies the actual address of the user's communication counterpart by analyzing, extracting, and comparing the features of the communication traffic obtained during monitoring.

In 1998, Cheng et al. [19] were the first to apply the website fingerprinting attack to traffic identification by using the feature of file size to identify some specific SSL-protected files. With the rise of anonymous networks, Herramnn et al. [17] in 2009 performed website fingerprinting to identify JAP, Tor, OpenSSH, OpenVPN, Stunnel, and CiscoIPsec-VPN. In 2011, Panchenko et al. [16] introduced a unique traffic burstiness combined with an SVM algorithm that achieved a 54% identification rate for Tor traffic. In subsequent studies, Wang et al. [10] extracted over 3000-dimensional feature vectors to model website fingerprinting and employed a weighted distance-based metric and a k-NN classifier to measure the similarity of website fingerprinting. Panchenko et al. [15] proposed the CUMUL method that exploited the feature of cumulative packet size, while Hayes et al. [8] proposed a random forest-based attack method (k-FP) to describe website fingerprinting by selecting 150 important features from the total 4,000 dimensions. Current methods are implemented by handcrafted feature sets combined with machine learning algorithms for website fingerprinting attacks and managing an accuracy exceeding 90%.

With the development of deep learning techniques in image, speech, and video, researchers have extended using deep learning schemes for website fingerprinting attacks. In 2016, Abe et al. [9] first succeeded using a Stacked Denoising Autoencoder (SDAE) for website fingerprinting attacks. In 2017, Rimmer et al. [7] extensively evaluated the performance of deep learning methods such as SDAE, CNN, and LSTM in a dataset consisting of 900 sites (each with 2500 samples). The reported results revealed that CNN provided the best results, achieving 96.66% accuracy in a closed world, while in an open environment, it achieves a TPR of 71.3% and an FPR of 3.4%. In 2018, Sirinam et al. [5] designed a more complex deep learning model (DF) with a deeper network structure that involves more convolutional and Batch Normalization layers. It eventually achieved 98.3% accuracy in a closed world consisting of 95 websites and 99% accuracy in an open world with 94% recall.

However, using deep learning for website fingerprinting attacks requires a large number of training samples per site. Hence, to solve this problem, Sirinam et al. [3] in 2019 first designed a Triplet Fingerprinting (TF) method for website fingerprinting attacks using a small-sample technique [18, 20–22], which involved a triplet network including an anchor (A), positive (P), and negative (N) as subunits of the triplet network. This method employs the cosine distance algorithm to measure the relationship between A-P and A-N, so that A and P are close to each other, while A and N are far away in the embedding space generated by the model. This means that the feature vectors generated by the same website sample traffic are close to each other, and the feature vectors generated by different website sample traffic are far apart. After training, the trained model is used as a feature extractor for website traffic, and then k-NN is used as a

classifier to finally achieve 95% accuracy requiring a small number of samples per website. Oh et al. [1], in 2021, first proposed another highly representative fingerprinting attack technique for low data sites, entitled GANDaLF, based on generative adversarial networks (GAN). This method uses a small number of labeled data and a more extensive unlabeled set to train the generator and the discriminator. The generator is trained to convert random seeds into pseudotraces with the same statistical distribution as the training data. The discriminator is trained to correctly exploit data for classification while distinguishing between the generator's true traces and pseudotraces output. This approach uses the generator as an additional data source to help improve the performance of the discriminator, making the website fingerprinting attack effective even in low data settings.

3. Attacks Based on Homology Analysis

3.1. Attack Threat Model. The website fingerprinting attack aims to disrupt the user's anonymity while visiting a website by utilizing traffic analysis; that is, the eavesdropper can infer the target websites visited by users from the encrypted anonymous traffic, with the primary attack model presented in Figure 1.

In this paper, we adopt the important assumptions of a website fingerprinting attack; that is, the attacker can only obtain the network packets on the communication link passively and cannot modify, delete, or insert any packet and encrypt, decrypt, or analyze the packets directly. The attacker collects the traffic, compares it with previously known traffic characteristics such as packet size, direction, and time interval, and finally finds the best match to the targeted website data stream record. In this way, the attacker is informed about the websites visited by a user and thus compromises the user's anonymity.

3.2. Website Fingerprinting Homology Analysis. The essence of the website fingerprinting attack is matching traffic characteristics, which is essentially the same goal as the homology detection of proteins and DNA in biology. Both scenarios aim to find similar segments between sequences, so we consider homology analysis feasible for the website fingerprinting attack. The homology analysis methods are commonly used in biology and are divided into three categories [23]: comparison-based, ranking-based, and discriminative-based methods. The most commonly used comparison-based methods are sequence, sequence spectrum, and HMM comparison, i.e., comparing sequences by dynamic programming and scoring functions. For example, in 2017, Zhuo et al. [6] implemented a website fingerprinting attack using a PHMM model. The core idea of the sorting-based approach is to regard homology detection as an information retrieval problem and sort the known sequences in the database and the unknown query sequences according to the homology relationship from near to far. The critical process of this method is the design of the sorting algorithm. According to the closeness of homology relationship, the discriminative-based approach involves dividing the

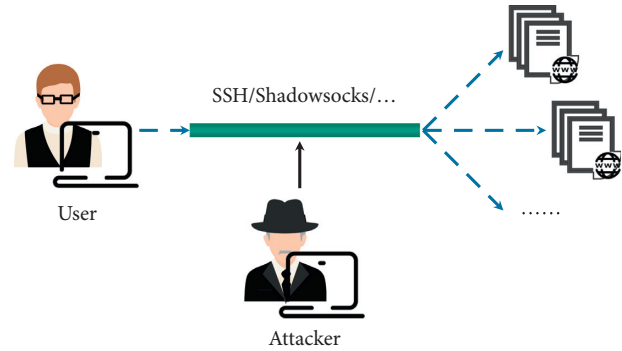


FIGURE 1: Threat structure model for website fingerprinting attacks.

sequences into positive and negative sample training and test sets. Then use the sequences in the training set to train the classification model based on machine learning and deep learning, and the test set evaluates the classifier's performance.

Traditional website fingerprinting attacks using deep neural networks require a large amount of data, and when the training data is insufficient, the model is less effective during classification. Additionally, the website content changes significantly over time, and these changes affect the website fingerprinting features. Therefore the model needs to be retrained after a while. At the same time, the migration ability of the model is weak, and the classification accuracy will drop significantly when the trained model is applied to a new classification task.

In this paper, we adopt a discriminative approach for website fingerprinting homology analysis. Unlike the traditional direct classification of website fingerprinting using machine learning and deep learning models, we adopt the structure of Siamese Networks [18]. During training, the purpose of our model is to change from directly attributing traffic sequences to corresponding website categories and train the network to learn the correlation between website traffic features, that is, the homology between website fingerprinting. This is achieved by using less data for model training to achieve a higher accuracy rate of website fingerprinting attacks.

3.2.1. Siamese Networks. Siamese Networks are a particular type of neural network structure, which, unlike a network model that learns to classify inputs directly, aim to learn the similarities and the correlations between the two inputs. The model selects the most likely identical category for a classification task by comparing each example in the test set with the training set. The Siamese Networks consider two samples on the input simultaneously and finally output the probability that they belong to the same category.

As shown in Figure 2, the Siamese Networks have two inputs X_1 and X_2 , in each cell structure, where X_1 and X_2 are input into the neural networks Network_1 and Network_2 with shared weights (in the usual case, it can be considered that Network_1 and Network_2 are two identical neural network structures). Then, a similarity measure algorithm is used to calculate the distance between the high-dimensional

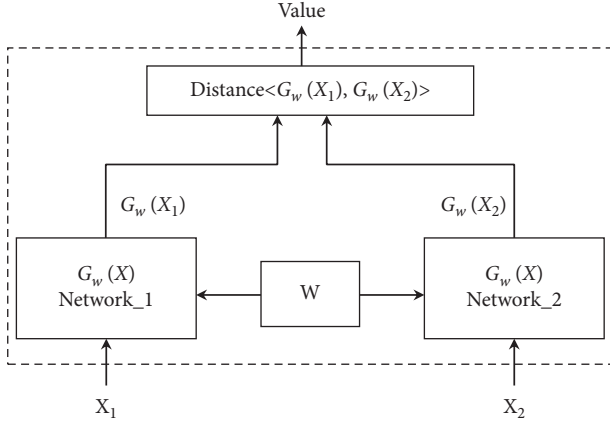


FIGURE 2: The structure model of Siamese Networks unit.

features $G_w(X_1)$ and $G_w(X_2)$ extracted by the neural network and the output value as the correlation measure of X_1 and X_2 .

The training test of the Siamese Networks contains multiple Siamese Network units, and each twin unit accepts two input data. Figure 3 illustrates the training test structure of the Siamese Networks, including the input, network, distance, and output layers. The input layer combines the input data, and the two inputs are logically symmetric because the network layer weights are shared, and the network structure is consistent. The network layer uses deep neural networks to extract high-dimensional features from the input data, commonly used as a CNN. The distance layer calculates the correlation between the high-dimensional features, and the typically used distance metrics are the cosine and sine. The output layer uses the results of the distance layer to get the probability that two inputs belong to the same category.

3.2.2. Dataset Construction Method. Each unit of a Siamese Network requires two inputs, and therefore the dataset needs to be correctly reconstructed. Assuming that N websites are to be classified, the training and test sets are defined by

$$\begin{cases} S_{\text{train}}(k) = S_{\text{train}}^+(k) \cup S_{\text{train}}^-(k) \\ S_{\text{test}}(k) = S_{\text{test}}^+(k) \cup S_{\text{test}}^-(k) \end{cases} \quad (k = 1, 2, 3, \dots, N), \quad (1)$$

where k denotes the dataset for the k -th website prediction classification, $S_{\text{train}}^+(k)$ and $S_{\text{train}}^-(k)$ are the positive and negative sample training set for the k -th website, respectively, and $S_{\text{train}}^+(k)$ and $S_{\text{train}}^-(k)$ together constitute the training set $S_{\text{train}}(k)$ for the k -th website. $S_{\text{test}}^+(k)$ denotes the positive sample test set of the k -th website, $S_{\text{test}}^-(k)$ denotes the negative sample test set of the k -th website, and $S_{\text{test}}^+(k)$ and $S_{\text{test}}^-(k)$ together constitute the test set $S_{\text{test}}(k)$ of the k -th website:

$$\begin{cases} S_{\text{train}}^+(k) = k_i^+ \cup k_j^+ \\ S_{\text{train}}^-(k) = k_i^+ \cup k_l^- \end{cases} \quad (1 \leq i < j \leq P, 1 \leq l \leq P). \quad (2)$$

We assume that each website provides P samples for model training (equation (2)), k_i^+ and k_j^+ denote any two training samples from the k -th website, and the two inputs of the Siamese Networks unit are logically symmetric. Then, $S_{\text{train}}^+(k) = k_i^+ \cup k_j^+$ denotes that $S_{\text{train}}^+(k)$ consists of any two training samples from the k -th website, with k_l^- referring to the training samples of other sites than the training samples of the k -th site. To balance the number of samples of $S_{\text{train}}^+(k)$ and $S_{\text{train}}^-(k)$ in the training set, we select only one random sample as k_l^- for each site other than the training samples of the k -th site, and $S_{\text{train}}^-(k) = k_i^+ \cup k_l^-$ indicates that the two combinations of k_i^+ and k_l^- together form a negative sample training set for the k -th website.

$$\begin{cases} S_{\text{test}}^+(k) = k_i^+ \cup k_j^+ \\ S_{\text{test}}^-(k) = k_i^+ \cup k_l^- \end{cases} \quad (1 \leq i < j \leq Q, 1 \leq l \leq Q). \quad (3)$$

We also assume that each site provides Q samples for model test evaluation (equation (3)), and then under the same principle, we obtain the positive sample test set $S_{\text{test}}^+(k)$ and the negative sample test set $S_{\text{test}}^-(k)$ for the k -th site.

3.3. CNN-BiLSTM-Based Siamese Networks Attack Model Construction

CNN. A convolutional neural network has four significant features, that is, the local perceptual domain, shared weights, pooling, and multilayer network, which can capture the complex features in the original data, and therefore it is widely used to process serial and image data. The original data is convolved with the local perceptual domain, and shared weights are utilized to form a feature map composed of local features. These are then passed through the pooling layer for integration and to perform data dimensionality reduction. The in-depth features involve high-dimensional, complex, and abstract features created after several convolutional and pooling layers. In previous studies [3, 5, 7], CNNs have been widely used as the dominant feature extraction method for website fingerprinting attacks.

LSTM. The long short-term memory network dynamically processes the input sequence according to the time series, and the output processed in the previous time step is used as the input on the next time step. At the same time, LSTM achieves the purpose of blocking irrelevant information, absorbing relevant information, and maintaining information in a cell state through the collaboration among input gates, forgetting gates, and output gates, which solves the problem of gradient disappearance and gradient explosion often encountered in the training process of recurrent neural networks (RNN). Therefore, LSTM is widely used in sequence information processing. The possibility of using LSTM for website fingerprinting attacks was also discussed in [8].

As shown in Figure 4, our deep learning architecture uses a combined network comprising a CNN and a

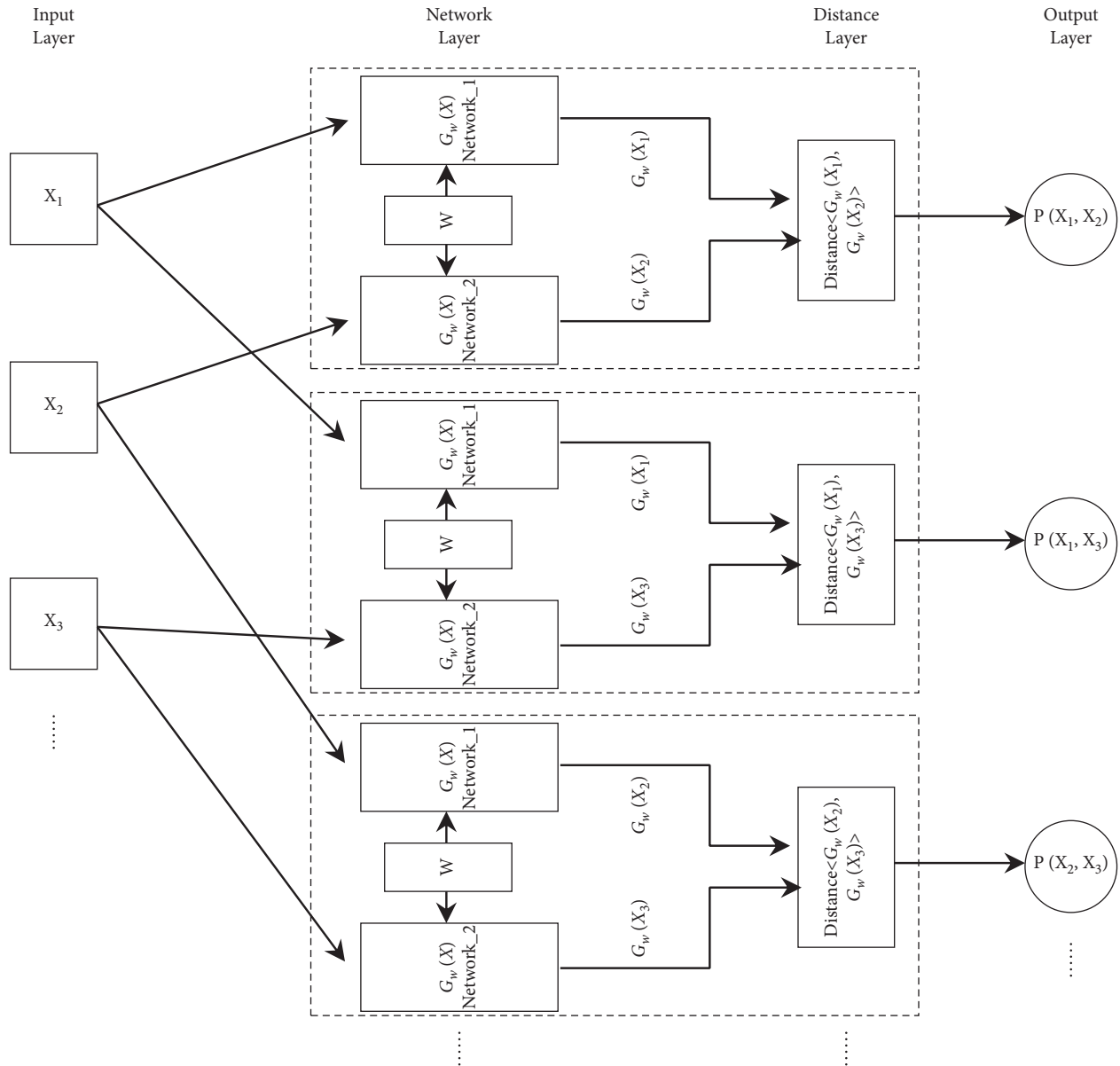


FIGURE 3: The structure model of Siamese Networks training and testing.

bidirectional LSTM (BiLSTM) as the base model of the Siamese Networks. Firstly, the CNN is used to extract the high-dimensional features of the two original input sequences, and then the dependencies in the high-dimensional features of the sequences are extracted through the BiLSTM layer. However, due to the long sequences generated by the network traffic, the output of LSTM at the last time step cannot represent the dependencies containing all subsequences, so we consider using the intermediate output of LSTM at each time step to better handle the local and global dependencies between the traffic sequences and the captured subsequences. At the same time, we choose a BiLSTM to replace the commonly used unidirectional LSTM. The forward LSTM in the BiLSTM model can extract the dependencies between the current input subsequence and its left subsequence, while the backward LSTM can extract the dependencies between the current input and its right

subsequence. Hence, the concatenation of these two intermediate outputs allows for more comprehensive information on the dependencies between the sequences. In the distance layer of twin networks, traditional distance measurement metrics such as cosine, sine, Euclidean, or other linear ones often underperform in evaluating the correlation between the high-dimensional features of the sequences. Thus, in this paper, we consider using fully connected neural networks as the distance measuring function. The features extracted from two original sequences are spliced, combined, and input to the fully connected layer to evaluate the homology relationship between the traffic sequences.

3.4. *Model Parameters.* To select the optimal hyperparameters for our model, we evaluate several CNN-BiLSTM model structures and parameters using the

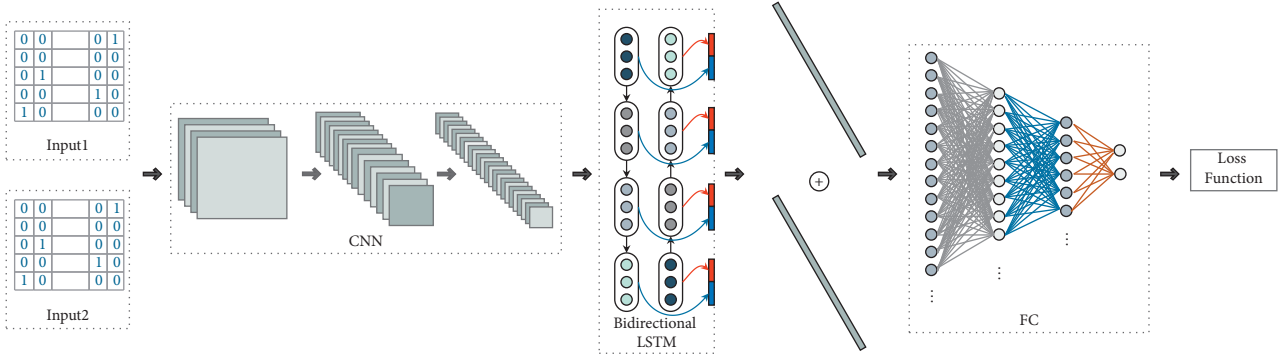


FIGURE 4: Structure of CNN-BiLSTM attack mode.

extensive candidate search method. Table 1 presents some of the critical parameter search spaces and the final selection.

We use Layer Normalization [24] for the Batch Normalization layer because the number of training samples we exploit is small, and Batch Normalization [24], which uses the mean and variance of the samples, does not reflect the global statistical distribution. Nevertheless, the Layer Normalization algorithm is independent of the batch size, and its statistics depend on the number of nodes in the hidden layer. For the network's activation function layer, we choose LeakyReLU [25], which presents the advantage of avoiding the neuron "death" faced by ReLU during training, reduces the parameters that need to be debugged, and improves training speed.

4. Dataset

4.1. Data Collection. The datasets used in this experiment are Liberatore's open dataset [26] and the Shadsocks [6]. As shown in Table 2, we also exploit two open datasets to construct the closed world and open-world datasets required for the experiment.

4.1.1. Closed World

SSH-200 Dataset. Built from the Liberatore open dataset, this dataset contains encrypted network traffic data from 2000 different sites accessed using SSH tunnels. However, this dataset involves many empty packets due to various failures during the collection process. For consistency, this experiment screens out sites with average instance SSH packet sequence length greater than 100 (based on the original dataset) and randomly selects 200 sites from them, with 25 instances selected for each site to generate the SSH-200 dataset.

Shadsocks-200 Dataset. 200 different domains were randomly selected from the top 1000 Alexa rankings, and each domain was accessed 25 times each using Shadsocks tunneling encryption to generate the Shadsocks-200 dataset.

4.1.2. Open World

SSH-2000 Dataset. One randomly selected instance from Liberatore's open dataset generates the SSH-2000 dataset for each site.

Shadsocks-2000 Dataset. It includes randomly 2000 selected websites from Alexa top 1000 to 10000 and uses Shadsocks tunnel to visit each website only once to generate Shadsocks-2000 dataset.

4.2. Data Processing. We process packets to filter out fragmented packets that do not provide reliable information in transmission, including missing, retransmitted, ACK loss, duplicate answers, and transmission packets with zero data segment length. Since the subject of this paper is SSH and Shadsocks anonymous network encrypted traffic without restrictions on the size of transmission units and packet delays like Tor [10], we extract the size, transmission direction, and time interval from each payload packet as the original sequence features.

This paper uses a one-hot matrix [23] to represent the original feature data, which requires sequence symbolization and construction of one-hot matrix processing for the original direction, size, and time interval feature sequences. After processing, we extend the feature dimension and the homology relationship between website fingerprinting features to enhance the measured feature distance.

4.2.1. Sequence Symbolization. Algorithm 1 describes the symbolization steps of the packet size and feature data direction, where the first two lines input the original packet sequence into the algorithm and extract them in order. Lines 3 to 7 merge the two attributes of size and direction, and lines 8 to 10 maximize the highlighted direction and size attributes in the form of double-symbol bits based on the maximum transmission unit MTU and the standard number of symbols Num. Finally, the double symbols are filled in cyclically to obtain the symbolized sequence S&D Seq.

Algorithm 2 describes the symbolization step of the packet time interval feature data with the input of the standard number of symbols Num and the maximum symbolization time interval Maxtime. The first two lines

TABLE 1: Model hyperparameter search space and final selection.

Hyperparameters	Search space	Selected value
Number of filters		
Conv2d1	[8 ... 32]	16
Conv2d2	[16 ... 64]	32
Normalization methods	[Batch Normalization, Layer Normalization]	Layer Normalization
Activation functions	[ReLU, ELU, LeakyReLU]	ReLU
Pooling layers	[Average, max]	Max
BiLSTM	[64 ... 256]	128
Number of FC layers	[1 ... 4]	3
[Filter, pool, stride] sizes	[2 ... 8]	[3, 3, 1]
Loss function	[Cross-entropy loss]	Cross-entropy loss
Optimizer	[SGD, adam, Adamax, RMSProp]	Adam
Learning rate	[0.0001 ... 0.01]	0.001
Training epochs	[10 ... 50]	30
Minibatch size	[16 ... 64]	48

TABLE 2: Dataset used in the experiment.

Name	Anonymous method	Training set	Test set	Purpose
SSH-200	SSH	200 × 20	200 × 5	Close world
Shadowsocks-200	Shadowsocks	200 × 20	200 × 5	Close world
SSH-2000	SSH	N/A	2000 × 1	Open world
Shadowsocks-2000	Shadowsocks	N/A	2000 × 1	Open world

Input: Packets Sequence Seq, Number of symbols Num
Output: Size and direction symbol sequences S&D Seq
Steps:
(1) S&DSeq ← Null
(2) **for** packet $a \in \text{Seq}$ **do**
(3) **if** $a.\text{Direction} = "+"$ **then**
(4) $a.\text{size} \leftarrow \text{MTU.size} + a.\text{size}$
(5) **else**
(6) $a.\text{size} \leftarrow \text{MTU.size} - a.\text{size}$
(7) **end if**
(8) $a.\text{S\&D.interval} \leftarrow (2 \times \text{MTU.size}) / \text{Num}^2$
(9) $a.\text{S\&D.symbol}[0] \leftarrow \text{Symbol}(a.\text{size} / (a.\text{S\&D.interval} \times \text{Num}))$
(10) $a.\text{S\&D.symbol}[0] \leftarrow \text{Symbol}(a.\text{size} \% (a.\text{S\&D.interval} \times \text{Num}))$
(11) S&D Seq.append($a.\text{S\&D.symbol}$)
(12) **end for**

ALGORITHM 1: Size and direction symbolization algorithm.

indicate that the original packet sequence is input to the algorithm, and the average symbolization time interval Time.interval is calculated based on the standard number of symbols Num and the maximum symbolization time interval Maxtime. The time of the first packet is also set as the base time. Lines 3 to 8 symbolize the time interval characteristics of the original packet sequence by first calculating the sequential two packet time interval ΔTime , which is set as a fixed character if the time interval ΔTime is greater than the maximum symbolization time interval. If the time interval ΔTime is less than the maximum symbolization time interval, each time interval Time.interval corresponds to a

symbol. Finally, the symbols are filled in cyclically to obtain the symbolization sequence $T\text{Seq}$.

4.2.2. Building a One-Hot Matrix. The original sequence is symbolized and can be expressed by

$$\text{Seq} = S_1, S_2, S_3, \dots, S_L, \quad (4)$$

where S_i denotes the i -th character of the symbolized sequence Seq and L denotes the length of the sequence. In this paper, the one-hot matrix, commonly used to represent DNA, RNA, and protein sequences in biology, represents the

Input: Packets Sequence Seq, Number of symbols Num, Max Time interval Maxtime,
Output: Time interval symbol sequences TSeq
Steps:
(1) TSeq \leftarrow Null, Time.base \leftarrow First packet.Time.now
(2) Time.interval \leftarrow Maxtime/Num
(3) **for** packet $a \in$ Seq **do**
(4) Δ Time \leftarrow a .Time.now – Time.base
(5) **if** Δ Time \geq Maxtime **then**
(6) a .Time.symbol \leftarrow Symbol(Max)
(7) **else**
(8) a .Time.symbol \leftarrow Symbol(Δ Time/Time.interval)
(9) **end if**
(10) Time.base \leftarrow a .Time.now
(11) TSeq.append(a .Time.symbol)
(12) **end for**
(13) **return** TSeq

ALGORITHM 2: Time interval symbolization algorithm.

symbolized sequences. For a sequence Seq, its one-hot matrix can be expressed as

$$M = \begin{bmatrix} e_{1,1} & \cdots & e_{1,L} \\ \vdots & \ddots & \vdots \\ e_{\text{num},1} & \cdots & e_{\text{num},L} \end{bmatrix}, \quad e_{i,j} = \begin{cases} 1, & S_j = \text{Symbol}_i, \\ 0, & \text{otherwise,} \end{cases} \quad (5)$$

where num denotes the number of standard characters and Symbol_{*i*} denotes the *i*-th standard character ($1 \leq i \leq \text{num}$). Intuitively, each character of the symbolized sequence can be represented by a num-dimensional vector, and only this character is activated in this vector. The value of this dimension is one, and the rest of the dimensions are zero.

To facilitate the training of the neural network model, we normalize the length *L* of the symbolized sequence. When the sequence length is greater than the preset normalized value *L*, we truncate the sequence, and if the length does not satisfy *L*, we complement it with zero (the num dimensional vector corresponding to zero in constructing a one-hot matrix is the zero-vector). Finally, all the original sequences are processed into num \times *L* matrices.

5. Experimental Evaluation

5.1. Assessment Indicators. To evaluate the experimental results, we use the following evaluation metrics: accuracy, true positive (TP), false positive (FP), true negative (TN), false positive (FP), precision, and recall. Accuracy indicates the ratio of the number of website categories correctly identified to the total number of websites in the same test set and is calculated by

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \times 100\%, \quad (6)$$

where TP is the number of monitored websites correctly classified, FP is the number of unmonitored websites incorrectly classified as monitored, TN is the number of unmonitored websites correctly classified, and FN is the

number of monitored websites incorrectly classified as different monitored or unmonitored websites. Recall refers to the percentage of monitored sites among the sites correctly classified by the classifier, and precision and recall are calculated by

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad (7)$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

5.2. Closed World Assessment. We evaluate the proposed model in the closed world case using SSH-200 and Shadowsocks-200 and demonstrate the parameter's interplay with the overall model's performance.

The accuracy of the model tested in the dataset SSH-200 is shown in Table 3. In a closed world and given some parameter setting conditions, our proposed CNN-BiLSTM model requires only 20 training samples and achieves up to 94.8% accuracy, performing significantly better than the traditional machine learning k-FP, k-NN, and PHMM models. Moreover, compared to the recently emerging small-sample website fingerprinting attack methods, the test results are slightly better overall than TF, the small-sample website fingerprinting attack model first proposed by Sirinam et al. in 2019 [3]. Additionally, our method's optimal test accuracy is equal to that of GANDaLF, the current state-of-the-art and data fingerprinting attack model proposed by Oh et al. [1].

In this section, we design comparative experiments to investigate the impact of using different combinations of traffic features and data representations on the accuracy of fingerprinting attacks. In the closed world, we employ the original direction and size features, that is, RawSize&Direction, and the original direction, size, and packet spacing combination features, that is, RawSize&Direction, Δ Time, the one-hot processed

TABLE 3: Test accuracy of dataset SSH-200 (%).

<i>Method name</i>	<i>Test methods</i>	<i>L = 200</i>	<i>L = 300</i>	<i>L = 400</i>	<i>L = 500</i>
CNN-BiLSTM	Raw Size&Direction	87.2	88.5	89.3	88.6
	Raw Size&Direction, Δ Time	85.6	87.9	88.3	89.7
	Directional Timing	86.9	88.9	90.2	89.8
	S&D Seq, one-hot	92.8	93.4	93.1	92.2
	S&D Seq, TSeq, one-hot	93.7	94.8	94.1	93.9
TF	S&D Seq, TSeq, one-hot	92.9	94.1	93.5	93.2
GANDaLF	S&D Seq, TSeq, one-hot	94.3	94.6	94.9	94.7
PHMM	S&D Seq, TSeq	85.9	87.3	88.2	86.5
<i>Method name</i>	<i>Test methods</i>			<i>K = 2</i>	<i>K = 3</i>
k-FP	S&D Seq, TSeq			91.2	91.1
k-NN	S&D Seq, TSeq			86.4	82.3

S&D Seq matrix, and the one-hot processed S&D Seq and TSeq combined matrix. Also, we compare our technique with the newly proposed directional timing-based attack (Tik-Tok attack) by Rahman et al. [2] in 2020. Table 3 highlights that the attack accuracy of the model can be improved by 4-5 percentage points using our proposed data representation technique compared with the direct use of raw traffic features and is significantly higher than the Tik-Tok approach using the combination of packet direction and timestamp features.

Meanwhile, we count the packet sequence lengths of the visited sites in the SSH-200 dataset. Figure 5 highlights that more than 75% of the sites have sequence lengths within 500, and thus, we set $L = 200, 300, 400,$ and 500 . It can be seen from Table 3 that the highest accuracy of the model classification, when tested directly using the original feature sequences of size and direction, is 89.3%, and the model classification accuracy decreases slightly because of the feature increment introduced in the dimension of the time interval. The latter is due to exploiting only 20 training samples and the subtle perturbation brought by the change of time interval affects the model's final training effect.

Additionally, due to the introduction of packet size and time symbolization interval, the original feature sequence after data processing presents for the same site multiple sample collections, imposing data changes in a particular range that does not change the symbol but improves the stability of the site fingerprint data features, making these statistical features uniquely representing a site. Therefore, after data processing, adding the dimensional feature of time interval improves the classification accuracy by 1.5%, and the model's highest attack accuracy is achieved at $L = 300$. Using the combined sequence of S&D Seq and TSeq after the one-hot matrix processing, the accuracy increases to 94.8%. The test results in Table 3 also indicate that, after data processing, as the normalized sequence length L increases, the model reaches the peak classification accuracy earlier. This is because the one-hot matrix introduces more zero elements in the vector while expanding the feature dimension, and the increase of the normalized sequence length L leads to more and more traffic sequences generated by the sites needing to be zero-complemented, making the sequences look more similar to each other after data processing.

The test results in Table 3 reveal that the highest classification accuracy is improved by nearly 5% after symbolizing the original feature data and constructing the one-hot matrix. We designed the following validation experiments to analyze the interplay between the number of standard symbols (packet size symbolization interval and time symbolization interval) and the accuracy during the symbolization process.

Figure 6 presents the model attack accuracy curves, where the number of standard symbols Num involves sequence lengths of $L = 200, 300,$ and 400 . It is clear that the accuracy rate keeps improving with the increase of Num (for $0 \leq \text{Num} \leq 20$), and the attack performance of the model reaches the optimum when the standard number of symbols is Num = 20. After that, the performance of the model starts to gradually decrease (for $\text{Num} \geq 20$). Hence, we conclude that the model's performance is related to the size of the symbolized interval division. When the standard number of symbols Num is small, the symbolization interval is large. The serialization process is more fault-tolerant to minor variations in packet size and time intervals in different samples from the same site. These features allow the model to categorize the samples originating from the same site, but a too-large interval will lead to the sequence not being obvious enough. The sequence generated by the samples of different sites varies less, which is not conducive to the differentiation of different sites, thus affecting the model's overall performance. When the number of standard symbols Num is larger, the symbolization interval is smaller. After symbolizing the original data, the samples from different sites will have apparent differences, which is beneficial to classify samples from different sites. However, for the different samples generated by multiple visits to the same site, the perturbations generated by the packet size and time interval change will show more apparent differences in their symbolization sequences, which is not conducive to the homology analysis. This is because samples from the same site will affect the classification ability of the model.

The tested accuracy of the CNN-BiLSTM model on the dataset Shadowsocks-200 is shown in Table 4. The model remains efficient in classifying and identifying Shadowsocks anonymous encrypted traffic, achieving a maximum attack accuracy of 98.1% with only 20 training samples per site when classifying against SSH anonymous encrypted traffic.

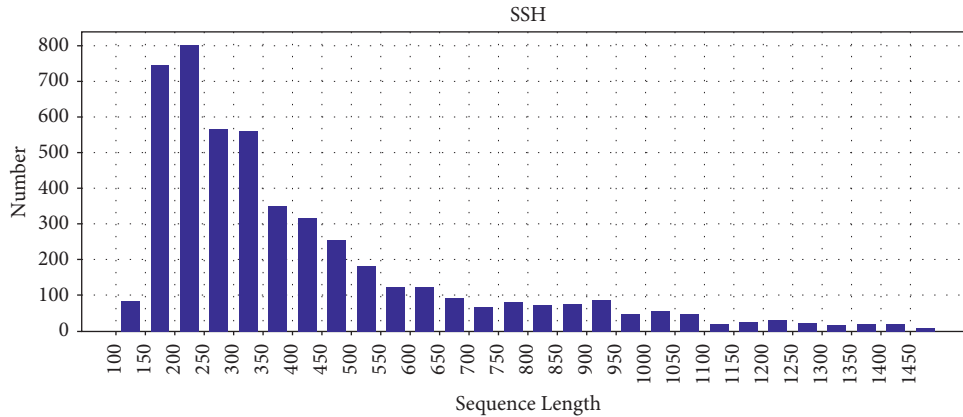


FIGURE 5: Statistics of site sequence length in SSH-200 dataset.

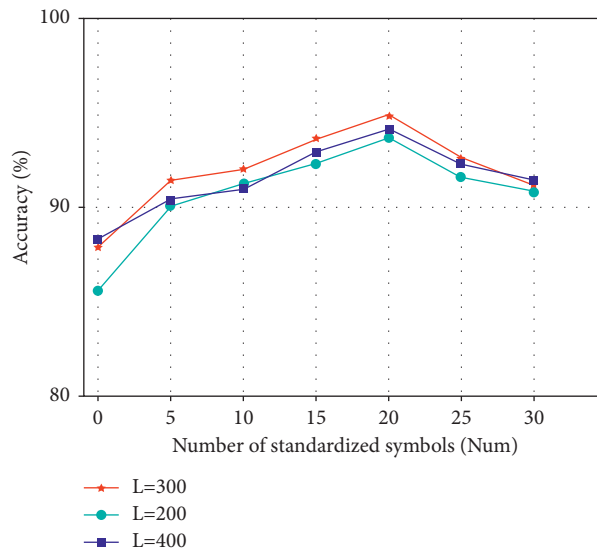


FIGURE 6: Curve of model accuracy with an increasing number of standard symbols.

TABLE 4: Test accuracy of dataset Shadsocks-200 (%).

Method name	$L = 200$	$L = 300$	$L = 400$
CNN-BiLSTM	97.6	98.1	96.3

This shows that each site’s packet direction, size, and time interval in the Shadsocks anonymous environment are more prominent, while each site’s traffic has fewer burst features and a smoother state, making it easier for eavesdroppers to perform website fingerprinting attacks.

5.3. Migration Capability Assessment. Transfer learning [27] is a deep learning-related technique, where an already trained CNN is partially retrained on an entirely new classification task. The performance of the newly trained model involves measuring its migration ability. Deep learning models can automatically extract data features from large amounts of data by semisupervised or unsupervised feature learning algorithms and hierarchical feature

extraction schemes and manage a higher classification accuracy than traditional machine learning methods. However, traditional website fingerprinting classification methods that employ deep learning, such as DF and AWF, require the training and test data to be independent and codistributed. If a model trained in the monitored website dataset collection A is used to classify fingerprint data in the untrained monitored website collection B, the attack accuracy of the deep learning model will drop drastically. Additionally, much time is required to collect the monitored website data in collection B and retrain the attack model, which is unacceptable to the attacker.

To evaluate the migration capability of the model, we consider a more challenging scenario and conduct experiments using the SSH-200 and Shadsocks-200 datasets.

SSH and Shadsocks are two completely different anonymous communication systems producing very different fingerprint data characteristics and collect significantly different site information. Our model is trained using one dataset, and the trained model is retrained by randomly selecting $R (R \leq 10)$ samples for each site in the other dataset, with the latter dataset also exploited as a testing dataset to evaluate the model's classification accuracy. Considering our trials, we evaluate the classification accuracy of the CNN-BiLSTM, TF, AWF, DF, and GAN-DaLF models with SSH anonymous fingerprint data as the training set and employ the Shadsocks anonymous fingerprint data as the test set. The corresponding results are illustrated in Figure 7.

As seen in Figure 7, the TF, GAN-DaLF, and CNN-BiLSTM models significantly outperform the traditional deep learning models. Since the test set and the training set are different types of traffic data, the data distribution is weakly correlated, and the trained model is directly applied to the classification task of the Shadsocks dataset. The accuracy of the traditional deep learning AWF, DF, and GAN-DaLF models based on the GAN network is less than 10%. In comparison, the attack accuracy of both TF and CNN-BiLSTM models exceeds 70%. As the number of samples (R) involved in the transfer learning process (secondary training) increases, the model's attack accuracy gradually improves with TF and CNN-BiLSTM's accuracy when $1 \leq R \leq 3$, but in principle, this improvement effect remains the same. The accuracy of TF and CNN-BiLSTM stabilizes above 90%, and when $R = 10$, the CNN-BiLSTM model accuracy is close to 92%, which is a 6% improvement over the TF method. The GAN-DaLF model has a significant improvement in attack accuracy as the sample number R increases due to its robust data generation capability, managing a close to the TF model performance for $R = 10$, and the accuracy curve still maintains a slow upward trend. The accuracy improvement effect of the traditional methods AWF and DF as the sample number R increases is more evident than TF and CNN-LSTM methods but much lower than GAN-DaLF model. The accuracy rate is already close to 50% at $R = 10$, but still, 40% lower compared with the CNN-LSTM method. This indicates that traditional deep learning models have limitations in adapting to new classification tasks and that CNN-LSTM, TF, and GAN-DaLF models can all better mitigate the adverse effects of data mismatch. However, the CNN-LSTM method has better migration ability in environments where samples are lacking.

5.4. Open-World Assessment. The performance of classifiers in the open world is another essential evaluation metric in website fingerprinting attacks. The goal is to assess the ability of the model to distinguish traffic generated by monitored websites from traffic generated by any other unknown websites. We use precision and recall to evaluate the CNN-BiLSTM model in an open-world scenario by plotting the precision-recall curve.

This section evaluates the model's performance in the SSH and Shadsocks anonymous communication systems.

To balance the number of monitored site samples with the number of monitored samples, we randomly select 10 samples for each site from the SSH-200 and Shadsocks-200 datasets to construct a monitored test sample set. The latter is then combined with the SSH-2000 and Shadsocks-2000 datasets to form the SSH and the Shadsocks open-world test set. At the same time, to better distinguish the monitored and unmonitored sites, we use the standard model during training and treat the unmonitored sites as an additional label.

Figure 8 presents the precision-recall curves of the CNN-BiLSTM model for sequence lengths of $L = 200, 300,$ and 400 in the SSH and Shadsocks open world. This figure highlights that the accuracy and recall rates are better in Shadsocks than in SSH, which indicates that the model is more suitable for Shadsocks' open-world environment for website fingerprinting attacks. As the recall rate increases, the classification accuracy rate significantly decreases for SSH and Shadsocks but is still between 0.7 and 0.8. Also, in both environments, the model performance is optimal for a sequence length of $L = 300$.

Under small-sample conditions, we further evaluate two extremely optimal models for website fingerprinting attacks in the open world: TF and GAN-DaLF. We test the performance of each model for sequence length $L = 300$ and plot the precision-recall curves with the corresponding results shown in Figure 9. All three models perform better in the open-world environment of Shadsocks, indicating that the individual characteristics of Shadsocks anonymous traffic data sites are more prominent and easier for model classification. The CNN-BiLSTM model performs significantly better than the TF model in both open-world environments. Furthermore, compared with the GAN-DaLF model in both open environments, each has its advantages and disadvantages.

The model's performance is appropriately optimized for precision or recall at $L = 200, 300,$ and 400 (Table 5). When the model is tuned for optimum precision rate, SSH reaches the highest precision rate of 0.889 at a sequence length of $L = 400$ with the corresponding recall rate being 0.831. Shadsocks reaches the highest precision rate of 0.912 at $L = 300$, with the recall rate being 0.899. Accordingly, when the model is optimized for the recall rate, both SSH and Shadsocks reach the highest performance at $L = 300$, managing the highest recall rates of 0.934 and 0.963, respectively, while the corresponding precision rates are 0.742 and 0.789.

Figure 8 and Table 5 highlight that the CNN-BiLSTM model is still highly usable in the open-world scenario, and the attacker can tune the model in the open world utilizing the task target. If the goal is to identify the traffic of monitored websites in the network data, then the recall rate should be of more concern to the attacker, and the accuracy rate can be appropriately sacrificed to improve the recall rate. Furthermore, when the attacker's goal is to accurately monitor the websites' visitors, the accuracy rate is more critical, and the recall rate needs to be appropriately reduced.

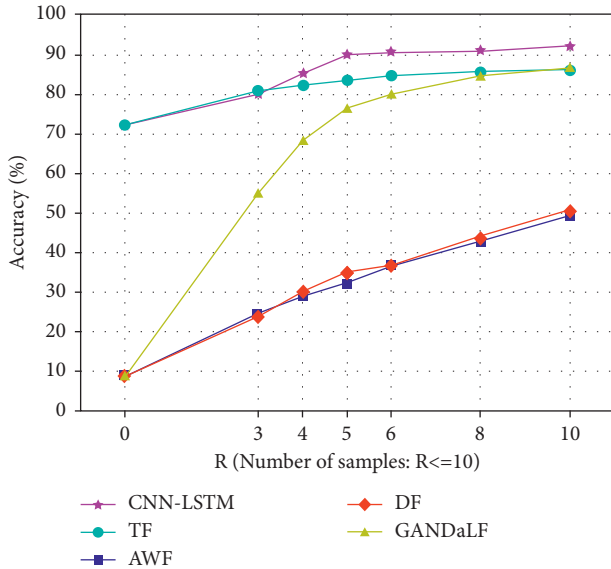


FIGURE 7: Accuracy curve with the number of “secondary training” samples.

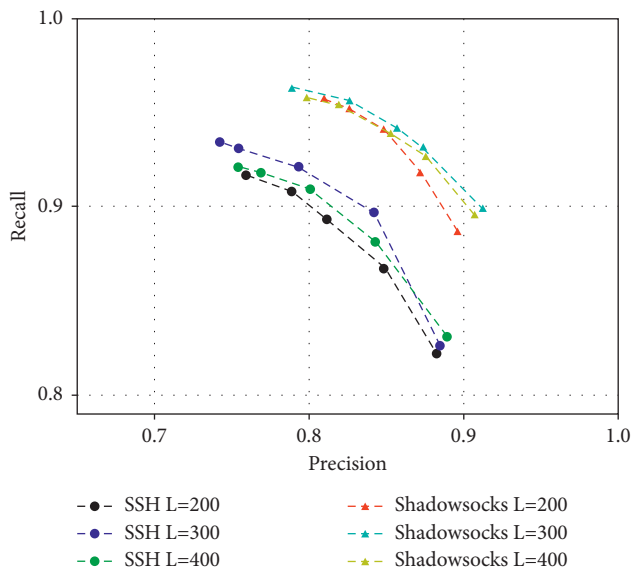


FIGURE 8: Precision-recall curves of the CNN-BiLSTM models in an open-world scenario.

6. Discussion

In this section, we discuss the possible limitations of this work and directions for future work.

6.1. Segmentation of Anonymized Web Data. In our experiments, we use previously collected representative datasets to ensure the purity of the data assuming that users open only one web page at a time during data collection. However, in a real-world attack scenario, users will open web pages accompanied by a lot of background traffic. Therefore, efficiently splitting the anonymous traffic from the background traffic is an important research topic.

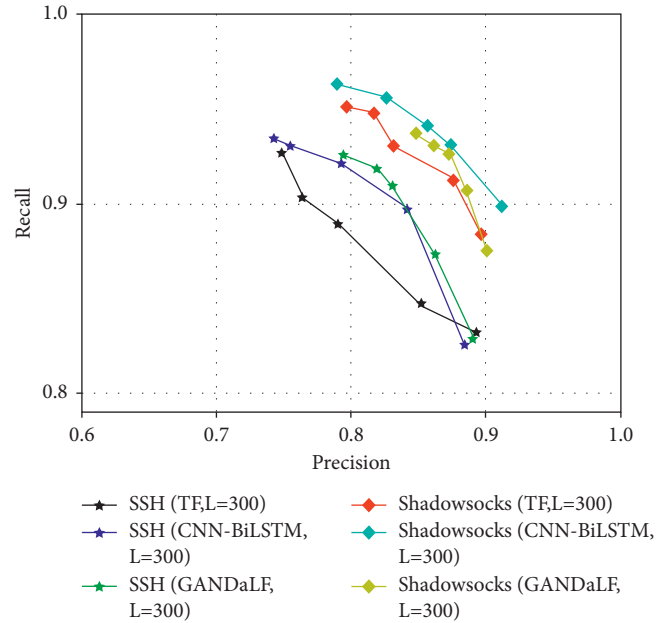


FIGURE 9: Precision-recall dynamic curves of CNN-BiLSTM, TF, GANDaLF models in the open world.

6.2. The Definition of Website Fingerprinting Attack. Our work is consistent with most current studies that only identify single-page website fingerprinting classification and do not include the hyperlinks and other subpages on the website homepage. The next step is to focus on how to characterize the overall fingerprint of the website.

6.3. Model Breakthroughs on Website Fingerprinting Defense Technology. This paper identifies and classifies the SSH and Shadowsocks single-agent anonymous encrypted traffic and employs the packet size, direction, and time interval as the essential features to achieve better attack results. To defend against website fingerprinting attack techniques that compromise user privacy, Tor, the currently best anonymous network communication system, was designed to transmit data in units in units of 512 bytes, called cells, and always pad all data transfers up to a cell boundary, with targeted defense against the important feature of packet size. Subsequent researchers have further defended against other features. Examples are the WTF-PAD based on adaptive padding proposed by Juarez et al. [28], Walkie-Talkie based on half-duplex communication and burst traffic proposed by Wang et al. [29] in 2017, Traffic Silver presented at USENIX Security 2020 proposed by Cadena et al. [30], zero-delay proposed by Gong and Wang et al. [31], and Mockingbird based on GAN techniques proposed by Rahman et al. [32]. These anonymity network defense techniques change the original direction, transmission time, and other characteristics of website traffic, blurring the differences between website traffic characteristics and increasing the difficulty for attackers to implement website fingerprinting attacks. Therefore, the model will have predictable degradation in attack effectiveness when applied to this more challenging anonymous network environment. A deeper analysis is

TABLE 5: Tuning results of CNN-BiLSTM model in the open world.

Environment	Length of sequence	Tuned for precision		Tuned for recall	
		Precision	Recall	Precision	Recall
SSH	$L = 200$	0.883	0.822	0.759	0.917
	$L = 300$	0.885	0.826	0.742	0.934
	$L = 400$	0.889	0.831	0.754	0.921
Shadowsocks	$L = 200$	0.896	0.887	0.809	0.957
	$L = 300$	0.912	0.899	0.789	0.963
	$L = 400$	0.907	0.896	0.798	0.958

needed on how to achieve a highly accurate small-sample website fingerprinting attack under such more complex conditions.

7. Conclusion

This paper proposes a website fingerprinting attack method based on homology analysis and designs a CNN-BiLSTM website fingerprinting attack model using a Siamese Network structure. Our architecture manages a high accuracy rate with only a small number of training samples per website. At the same time, we innovatively propose a data processing method to increase the data feature dimension and increase the fault tolerance of the sample's burst features.

We train our model with SSH anonymous network encrypted traffic and then exploit it to classify the Shadowsocks anonymous network encrypted traffic, managing over 90% accuracy with only five samples per site, which is significantly higher than current methods. Additionally, this experimental setup (training versus testing datasets are of different nature) highlights that the proposed model has a very appealing migration capability. Finally, the experimental results indicate that attackers can still achieve effective website fingerprinting attacks with fewer resources.

Data Availability

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

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The National Key Research Projects supported this topic (nos. 2019QY1302 and 2019QY1305). The authors would like to express their gratitude to EditSprings (<https://www.editsprings.com/>) for the expert linguistic services provided.

References

- [1] S. E. Oh, N. Mathews, M. S. Rahman, M. Wright, and N. Hopper, "GANDaLF: GAN for data-limited fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 2, pp. 305–322, 2021.
- [2] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, "The utility of packet timing in website fingerprinting attacks," vol. 2020, no. 3, pp. 5–24, 2020, <https://arxiv.org/abs/1902.06421>.
- [3] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: more practical and portable website fingerprinting with N-shot learning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1131–1148, London, United Kingdom, November 2019.
- [4] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-CNN: a data-efficient website fingerprinting attack based on deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 292–310, 2019.
- [5] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: undermining website fingerprinting defenses with deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1928–1943, Toronto, Canada, October 2018.
- [6] Z. Zhuo, Y. Zhang, Z. L. Zhang, X. Zhang, and J. Zhang, "Website fingerprinting attack on anonymity networks based on profile hidden Markov model," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1081–1095, 2017.
- [7] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, *Automated Website Fingerprinting through Deep Learning*, arXiv, preprint arXiv:1708.06376, 2017.
- [8] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 1187–1203, Austin TX USA, August 2016.
- [9] K. Abe and S. Goto, "Fingerprinting attack on Tor anonymity using deep learning," *Proceedings of the Asia-Pacific Advanced Network*, vol. 42, pp. 15–20, 2016.
- [10] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pp. 201–212, Berlin Germany, November 2013.
- [11] J. Cheng, Y. Li, C. Huang, A. Yu, and T. Zhang, "ACER: detecting Shadowsocks server based on active probe technology," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 3, pp. 217–227, 2020.
- [12] B. Zantout and R. Haraty, "I2P data communication system," in *Proceedings of ICN*, pp. 401–409, ICN, Springer, Singapore, 2011.
- [13] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Proceedings of the The 13th conference on USENIX Security Symposium*, CA, USA, August 2004.
- [14] J. Boyan, "The anonymizer-protecting user privacy on the web," *Computer-Mediated Communication Magazine*, vol. 4, no. 9, 1997.

- [15] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, and M. Henze, *Website fingerprinting at internet scale*, NDSS, NY, USA, 2016.
- [16] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pp. 103–114, Chicago Illinois USA, October 2011.
- [17] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 31–42, 2009.
- [18] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proceedings of the ICML deep learning workshop*, Lille, France, July 2015.
- [19] H. Cheng and R. Avnur, *Traffic Analysis of Ssl Encrypted Web Browsing*, University of Berkeley, CA, USA, 1998.
- [20] C. Zhang, Y. Cai, G. Lin, and C. Shen, "DeepEMD: few-shot image classification with differentiable earth mover's distance and structured classifiers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Article ID 12203, Seattle, WA, USA, June 2020.
- [21] W. Li, L. Wang, J. Xu, J. Huo, Y. Gao, and J. Luo, "Revisiting local descriptor based image-to-class measure for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7260–7268, Long Beach, CA, USA, June 2019.
- [22] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, pp. 4077–4087, MIT Press, Cambridge, MA, USA, 2017.
- [23] S. Li, J. Chen, and B. Liu, "Protein remote homology detection based on bidirectional long short-term memory," *BMC Bioinformatics*, vol. 18, no. 1, pp. 443–448, 2017.
- [24] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International conference on machine learning*, pp. 448–456, Lille, France, July 2015.
- [25] X. Zhang, Y. Zou, and W. Shi, "Dilated convolution neural network with LeakyReLU for environmental sound classification," in *Proceedings of the 2017 22nd International Conference on Digital Signal Processing (DSP)*, pp. 1–5, London, UK, August 2017.
- [26] Q. Sun, D. R. Simon, Y. M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 19–30, Berkeley, CA, USA, May 2002.
- [27] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pp. 242–264, IGI global, PA, USA, 2010.
- [28] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proceedings of the Computer Security - ESORICS 2016 European Symposium on Research in Computer Security*, pp. 27–46, Guildford, UK, September, 2016.
- [29] T. Wang and I. Goldberg, "Walkie-talkie: an efficient defense against passive website fingerprinting attacks," in *Proceedings of the 26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1375–1390, Vancouver BC, Canada, August 2017.
- [30] W. D. L. Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, and J. Filter, "Trafficliver: Fighting website fingerprinting attacks with traffic splitting," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1971–1985, Virtual Event USA, November 2020.
- [31] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *Proceedings of the 29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 717–734, Boston, MA, USA, August 2020.
- [32] M. S. Rahman, M. Imani, N. Mathews, and M. Wright, "Mockingbird: defending against deep-learning-based website fingerprinting attacks with adversarial traces," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1594–1609, 2020.

Review Article

AGG: A Novel Intelligent Network Traffic Prediction Method Based on Joint Attention and GCN-GRU

Huafeng Shi ^{1,2}, Chengsheng Pan ¹, Li Yang ¹, and Xiangxiang Gu ²

¹School of Automation, Nanjing University of Science and Technology, Nanjing 210094, China

²School of Information Engineering, Dalian University, Dalian 116622, China

Correspondence should be addressed to Huafeng Shi; shihuaifeng314@126.com

Received 2 July 2021; Accepted 14 September 2021; Published 30 September 2021

Academic Editor: Jinwei Wang

Copyright © 2021 Huafeng Shi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Timely and accurate network traffic prediction is a necessary means to realize network intelligent management and control. However, this work is still challenging considering the complex temporal and spatial dependence between network traffic. In terms of spatial dimension, links connect different nodes, and the network traffic flowing through different nodes has a specific correlation. In terms of temporal dimension, not only the network traffic at adjacent time points is correlated, but also the importance of distant time points is not necessarily less than the nearest time point. In this paper, we propose a novel intelligent network traffic prediction method based on joint attention and GCN-GRU (AGG). The AGG model uses GCN to capture the spatial features of traffic, GRU to capture the temporal features of traffic, and attention mechanism to capture the importance of different temporal features, so as to realize the comprehensive consideration of the spatial-temporal correlation of network traffic. The experimental results on an actual dataset show that, compared with other baseline models, the AGG model has the best performance in experimental indicators, such as root mean square error (RMSE), mean absolute error (MAE), accuracy (ACC), determination coefficient (R^2), and explained variance score (EVS), and has the ability of long-term prediction.

1. Introduction

Cisco annual Internet report (2018–2023) notes that device functionality will be combined with higher bandwidth and more intelligent networks by 2023, and the number of devices linked to IP networks will be more than three times the global population [1]. With the increasing number of terminals, the enrichment of multimedia applications, and the continuous expansion of network capabilities, network traffic management has become a critical and challenging task. Real-time and accurate network traffic prediction can greatly improve the control gain of the network.

The existing network traffic prediction methods are divided into model-driven traffic prediction methods and data-driven traffic prediction methods. Model-driven traffic prediction methods are also called parameterization methods, including autoregressive moving average model (ARMA) and autoregressive integrated moving average mode (ARIMA). Laner et al. introduced the ARMA model,

which can predict network traffic [2]. Guo et al. introduced the ARIMA model and tested the algorithm with the data collected by a backbone switching node. The experimental results show that compared with other network traffic prediction methods, the model has a better effect in dealing with nonstationary series and higher prediction accuracy [3], so the ARIMA model and its variants are widely used and can well explore the time correlation of network traffic [4–6]. Model-driven traffic prediction methods mostly use a polynomial fitting function to approximate the actual network traffic and then make the fitting effect better through a large number of parameter tuning. However, it is difficult to capture the nonlinear characteristics of network traffic, such as fast fluctuation and time dependence.

The data-driven traffic prediction method can automatically learn statistical rules from a large quantity of historical data to intelligently capture the nonlinear characteristics of network traffic. Specifically, data-driven traffic prediction methods can be divided into machine learning

prediction methods and deep learning prediction methods. Among them, machine learning prediction methods include support vector regression (SVR) and k-nearest neighbor algorithm (k-NN). Bermolen et al. applied support vector regression (SVR) to link load prediction [7]. Kremer et al. chose two different machine learning algorithms, SVR and KNN, to explore the balance between complexity and estimation accuracy [8]. However, machine learning methods are not sufficient for processing high-dimensional data and rely on feature engineering. Therefore, the universality of this method is weak.

Compared with machine learning prediction methods, deep learning prediction methods can not only retain the learning characteristics but also ensure the relevance between tasks and effectively address time series problems. Wu et al. proposed a network traffic prediction method based on a deep neural network (DNN), which proves the superiority of the deep learning prediction method in traffic prediction [9]. Lazaris et al. used actual network traffic tracking from ISPs to train long-term short-term memory (LSTM) neural network and generate predictions in a short time. Experiments show that LSTM can predict network traffic with low error [10]. Azzouni et al. proposed an LSTM RNN framework for predicting a large-scale network traffic matrix and proved the fast convergence ability of the LSTM model through actual data from GEANT [11]. Although this kind of deep learning prediction model has achieved good results, the above models all predict the time series of network traffic in a single area but ignore the spatial structure of the network, that is, the spatial correlation of network traffic. To extract the spatial characteristics of network traffic, researchers introduced convolutional neural networks (CNNs) into the task of network traffic prediction. Zhang et al. used a convolutional neural network to capture the temporal and spatial dependence of traffic by processing traffic data to images. The experimental results show that the prediction performance of this method in terms of root mean square error (RMSE) is significantly improved [12]. Li et al. proposed a CNN fusion LSTM model for prediction, used a one-dimensional CNN to obtain the spatial characteristics of network traffic, and used LSTM to obtain the temporal correlation of network traffic. However, the spatial structure of the CNN model is in Euclidean space; that is, the CNN can only deal with Euclidean data, but it cannot effectively deal with non-Euclidean data such as communication network topology.

Therefore, researchers hope to effectively extract spatial features from non-Euclidean data structures such as topological maps [13], so GCNs have become a new research focus. He et al. proposed a spatial-temporal network based on graph attention, which is called GSATN. This model integrates spatial-temporal characteristics, characterizes spatial correlation through geographical relationship graphs, characterizes temporal correlation through recurrent neural networks, and predicts network traffic by combining spatiotemporal characteristics [14]. Yang et al. proposed a network traffic prediction model combining a graph convolution neural network (GCN) and a gate control recursive unit (GRU). The model uses GCN to learn network topology

and extract spatial characteristics of traffic and uses GRU to learn the temporal characteristics of network traffic. Thus, the intelligent prediction of network traffic is realized [15]. Although these models have achieved excellent prediction accuracy, most models tend to extract static spatial dependencies in traffic, and such spatial dependencies may evolve over time [16, 17]. Therefore, by introducing an attention mechanism into the GCN-GRU model, this paper proposes a novel intelligent network traffic prediction method based on joint attention and GCN-GRU. This model can not only capture spatial-temporal correlation information but also collect temporal global change information. The main contributions of this paper are as follows:

- (1) A network traffic prediction method combining GCN, GRU, and attention mechanism is proposed. The method uses GCN to capture the spatial features of traffic, GRU to capture the temporal features of traffic, and attention mechanism to capture the importance of different temporal features, so as to realize the comprehensive consideration of the spatial-temporal correlation of network traffic.
- (2) The attention mechanism is introduced into the GRU, and the weight matrix calculation method in the GRU unit is redesigned. In this mechanism, the state vector is generated by combining the hidden states at different times, a scoring function is designed to calculate the weight of each hidden state, and an attention function is designed to calculate the context vector that can describe the global traffic change information, so as to adjust the importance of different time points and collect the global time information to improve the prediction accuracy.
- (3) Considering that the length of the sliding window and the number of hidden units have a significant impact on the timeliness and accuracy of network traffic prediction, an action to determine the experimental parameters is performed, so as to obtain the optimal length of sliding window and optimal number of hidden units, which effectively supports the comparative analysis of the network traffic prediction model AGG proposed in this paper with other baseline models.
- (4) The AGG model is trained on the Milan traffic network dataset for many times. The results show that compared with several existing baseline models, the AGG model has the best performance in experimental indicators, such as root mean square error (RMSE), mean absolute error (MAE), accuracy (ACC), determination coefficient (R^2), explained variance score (EVS), and has the ability of long-term prediction.

The rest of this paper is organized as follows. In Section 2, we present the problem formulation of network traffic prediction and design a framework to solve the network traffic prediction problem. Based on the design of the spatial feature extraction model, temporal feature extraction model, and attention mechanism model, a complete intelligent

network traffic prediction model is given in Section 3. In Section 4, we introduce the experimental environment and analyze the performance of the proposed traffic prediction model. We conclude this paper in Section 5.

2. The Proposed Prediction Framework

2.1. Problem Formulation. The goal of network traffic prediction is to predict the network traffic information in the future according to the measured historical network traffic information. We can define this process as

$$x_{t-M+1}, \dots, x_{t-1}, x_t \xrightarrow{f(\cdot)} (\cdot) \hat{x}_{t+1}, \dots, \hat{x}_{t+H-1}, \hat{x}_{t+H}, \quad (1)$$

where $x_t \in R^n$ is the observation vector of n observation points at the sampling time t . The purpose of the traffic prediction model is to learn a mapping function $f(\cdot)$ based on the traffic data of the previous M sampling time to predict the network traffic of the H sampling time in the future.

Definition 1 (network topology). The network is composed of nodes and links, which are generally represented by digraphs $G = (V, E)$. V represents the nodes in the network, and $V = \{V_1, V_2, \dots, V_N\}$, where N is the number of nodes, and E represents the links between nodes. The adjacency matrix A is used to represent the connection relationship of nodes, $A \in R^{N \times N}$. The adjacency matrix only contains the elements 0 and 1. When the element is 0, there is no connection between nodes, and when the element is 1, there is a connection between nodes.

Definition 2 (network traffic prediction). In G , each link is $e_i (1 \leq i \leq n)$, and the time series $x_{t-n}, \dots, x_{t-1}, x_t$ represents the network traffic of e_i in the time interval N . The principle of the prediction model proposed in this paper is to learn a mapping function f based on the topological graph structure and network traffic time series to obtain the network traffic data spatial-temporal characteristics and then predict the network traffic information x_{t+1}, \dots, x_{t+T} in the future from the characteristic matrix. The network traffic prediction formula is as follows:

$$[x_{t+1}, \dots, x_{t+T}] = f(G, (x_{t-n+1}, \dots, x_{t-1}, x_t)). \quad (2)$$

2.2. Traffic Prediction Framework. For the problem described in Section 2.1, the prediction architecture proposed in this paper is shown in Figure 1. First, the time series data in each region in the dataset at n time sampling points and the adjacency matrix representing the relationship between regions are taken as the input. Then, the GCN model is used to extract the input data spatial features, and the time series with spatial features are used as the input of the GRU model to extract the temporal correlation features between time series. Furthermore, the attention mechanism is introduced into GRU, and the weight matrix calculation method in the original GRU unit is replaced by the attention weight mechanism, which reweights the influence of historical network traffic data to capture the global variation trend of

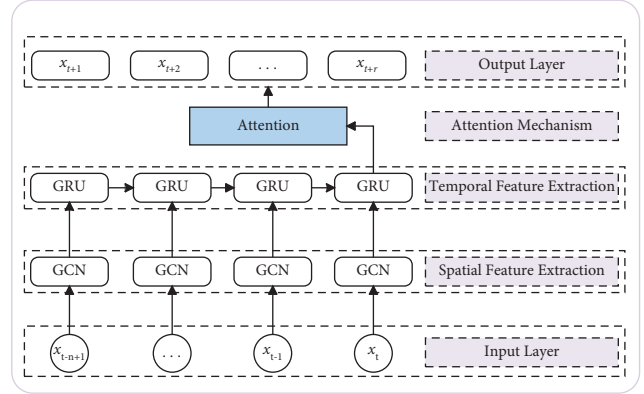


FIGURE 1: AGG prediction architecture.

network traffic. Finally, the prediction results of data with spatial-temporal correlation are obtained through the fully connected layer.

3. Prediction Models

3.1. Spatial Feature Extraction Model. Spatial feature extraction is one of the critical problems in network traffic prediction. A regional topological network is a graph structure, and its network traffic data belong to non-Euclidean data. Although traditional convolutional neural networks (CNNs) can obtain spatial features, they can only be used in Euclidean data and cannot effectively extract spatial features from graph data. In this paper, the graph convolution network (GCN) model is used to process the non-Euclidean data represented by graph data, and the spatial features of each region are learned from the network structure.

The principle of GCN is to construct a filter in the Fourier domain and then process the graph nodes and the first-order domain of the nodes with the constructed filter to obtain the spatial features between the nodes in the graph. Finally, the GCN model is established by superposition of multiple convolution layers. In this paper, we designed two convolutional layer processing graph structures, and the formula is as follows:

$$f(X, A) = \sigma(\text{ARReLU}(AXW_0)W_1), \quad (3)$$

where X represents the network traffic characteristic matrix, A represents the adjacency matrix, $\sigma(\cdot)$ and ReLU represent the activation function. $\hat{A} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$ represents the preprocessing step, \tilde{D} is the degree matrix, $\tilde{D} = \sum_i \tilde{A}_{ij}$. $\tilde{A} = A + I_N$ represents the matrix with a self-connection structure, and W_0 and W_1 represent the weight matrix in the first and second convolution layers, respectively.

3.2. Temporal Feature Extraction Model. Temporal feature extraction is another critical problem in network traffic prediction. At present, the recurrent neural network (RNN) is the most widely used neural network model for processing sequence data. However, due to the defects of gradient disappearance and gradient explosion, the traditional

recurrent neural network has limitations in terms of long-term prediction. The LSTM model and GRU model are variants of recurrent neural networks, which can better solve the above defects. As variants of RNN, LSTM, and GRU have the same basic principle, they both use a gate control mechanism to memorize as much long-term information as possible. In this paper, we use the GRU network unit. Compared with the LSTM unit, the GRU unit has fewer parameters. Under the premise of ensuring the prediction accuracy, it can reduce the time of model optimization.

The structure diagram of the GRU unit is shown in Figure 2, in which x_t represents the input data at time t , h_t , h_{t-1} , and h_{t+1} indicate the hidden state at different times, r_t is a reset gate, which controls the degree of information reservation or abandonment at the previous time, u_t is an update gate, which is used to control the extent to which state information of the prior moment enters the current state, c_t is the information stored at time t , and the principle of GRU is to use the hidden state of the prior moment and the input of the current moment together to obtain the network state information of the next moment. The model not only captures the current network information but also retains the change trend of historical network information and has the ability to capture temporal dependence.

3.3. Attention Mechanism Model. When capturing temporal features, we introduce an attention mechanism into GRU in this section and redesigns the weight matrix calculation method in the original GRU unit with the attention weight mechanism.

After replacing the original matrix calculation method in GRU with an attention mechanism, X_t and h_{t-1} are used to obtain the information of the reset gate r_t and update gate u_t at time t . The formulas are as follows:

$$\begin{aligned} r_t &= \sigma(W^k [X_t, h_{t-1} + b_r]), \\ u_t &= \sigma(W^k [X_t, h_{t-1} + b_u]), \end{aligned} \quad (4)$$

where W^k is the weight matrix information in the attention mechanism, X_t represents the input traffic at the current time, h_{t-1} represents the hidden state passed down from the previous time, and b_r and b_u are deviation parameters.

After obtaining the information of the reset gate r_t and update gate u_t , the reset data $h_{t-1}' = r_t \odot h_{t-1}$ can be obtained first, and then the value range of the data of h_{t-1}' and X_t can be controlled within $[-1, 1]$ through the tanh activation function. That is, the state of memorizing the current moment h' can be obtained. The formula is as follows:

$$h' = \tanh(W^k [X_t, (r_t \odot h_{t-1})] + b_n), \quad (5)$$

where b_u is a deviation parameter.

After obtaining the current time state of memory, the last step is to update the memory stage, in which the update gate u_t is used. The formula is as follows:

$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot h'. \quad (6)$$

Through the multilayer GRU with attention mechanism, the temporal features of network traffic can be better

captured. The internal structure of the redesigned GRU is shown in Figure 3.

3.4. Traffic Prediction Model. The network traffic prediction model, named AGG model, introduces the attention mechanism based on the GCN-GRU model and reweights the influence of historical network traffic data to capture the global variation trend in network traffic. The model structure is shown in Figure 4.

The AGG model calculation is shown in the following formulas:

$$\begin{aligned} u_t &= \sigma(W_u [GC(A, X_t), h_{t-1}] + b_u), \\ r_t &= \sigma(W_r [GC(A, X_t), h_{t-1}] + b_r), \\ c_t &= \tanh(W_c [GC(A, X_t), (r_t * h_{t-1})] + b_c), \\ h_t &= u_t * h_{t-1} + (1 - u_t) * c_t, \end{aligned} \quad (7)$$

where u_t is the update gate which is used to control the extent to which the state information of the last time enters the state of current time, σ is the activation function of the nonlinear model, W_u , W_r , and W_c are the weight parameters, GC is the graph convolution process, A is the adjacency matrix, X_t is the input of the model at the current time, h_{t-1} and h_t are the hidden state at $t-1$ and t , respectively, b_u , b_r , and b_c are deviation parameters, r_t is the reset gate which controls the level of information retention or abandonment at the previous time, and c_t is the information stored at time t .

The AGG model is constructed by the GCN model combined with the GRU model. The principle is to input n historical time series network traffic data into the AGG model to obtain n hidden states and obtain the vector containing spatial-temporal features: $\{h_{t-n+1}, \dots, h_{t-1}, h_t\}$.

Then, the hidden state is inputted into the attention model, and the multilayer perceptron (MLP) is used to calculate the weight of each hidden state h : $\{a_{t-n+1}, \dots, a_{t-1}, a_t\}$. The information vector covering the global traffic change is calculated by the sum of the weights. The formulas are as follows:

$$a_i = \frac{\exp(e_i)}{\sum_{k=1}^n \exp(e_k)}, \quad (8)$$

$$e_i = W_{(2)}(W_{(1)}H + b_{(1)}) + b_{(2)}.$$

Then, an attention function is used to describe the vector C_t of global traffic change information, and the formula is as follows:

$$C_t = \sum_{i=1}^n a_i * h_i. \quad (9)$$

Finally, the final predicted value is obtained through the fully connected layer.

4. Simulation Results and Analysis

In this part, we first introduce the actual traffic dataset of the telephone service provider in the European city of

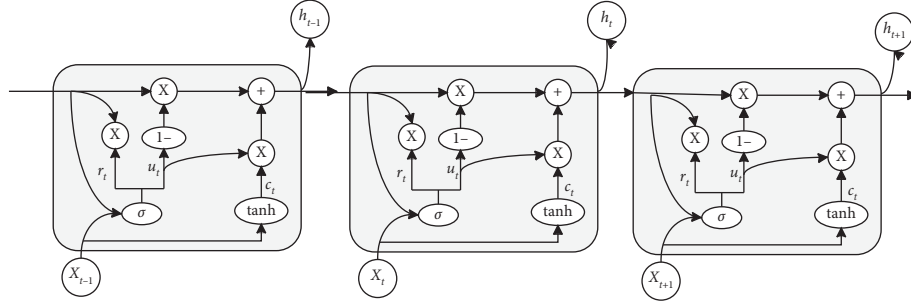


FIGURE 2: Schematic diagram of the GRU structure.

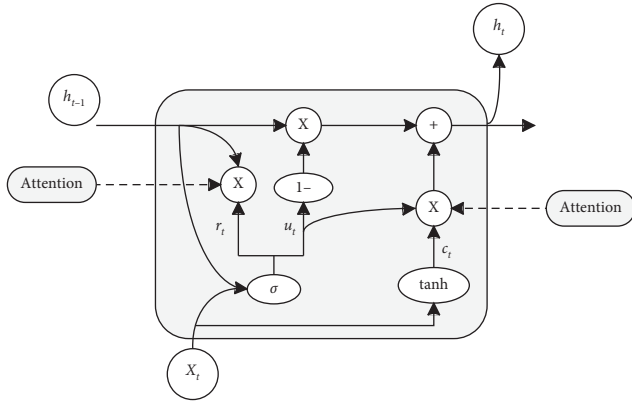


FIGURE 3: Internal structure of the GRU after redesign.

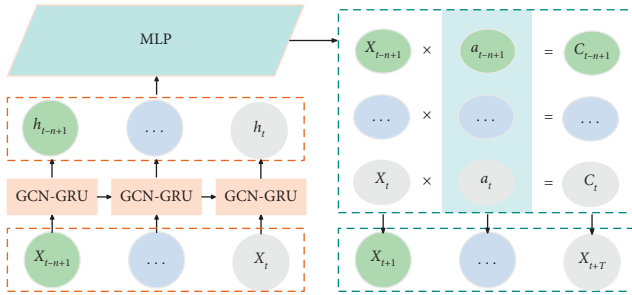


FIGURE 4: AGG model.

Milan and then analyze comparative experiments based on this dataset to verify the advantages of our proposed model.

4.1. Dataset Description. In this paper, we select an open network traffic dataset which is in <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EGZHFV>, and the traffic collection time is from 00:00 on November 1, 2013, to 00:00 on January 1, 2014. Table 1 shows the relevant dataset information. In this experiment, the data of 11/04-11/10 for seven days are selected as the dataset. The time interval of the original data is 10 minutes, and there are 144 data points in each region. In this paper, nine regions are selected, and the data of a week are collected. The grid and map of the area where the dataset is located are shown in Figure 5. Figure 6 shows the network traffic trend of the nine regions within a week.

4.2. Experimental Indicators. In order to thoroughly verify the performance of the model, we set five experimental indicators to judge the flow prediction model proposed in this paper, as follows:

- (1) Root mean square error (RMSE) reflects the prediction error of the model. The value range of RMSE is $[0, +\infty)$. The closer the RMSE is to zero, the better the performance of the model is.

$$\text{RMSE} = \sqrt{\frac{1}{T} \sum_{t=1}^T (Y^t - \hat{Y}^t)^2}. \quad (10)$$

- (2) Mean absolute error (MAE) is used to measure the mean absolute error between the predicted value and the true value. The value range of MAE is $[0, +\infty)$. The closer the MAE is to zero, the better the performance of the model is.

$$\text{MAE} = \frac{1}{T} \sum_{t=1}^T |Y^t - \hat{Y}^t|. \quad (11)$$

- (3) Accuracy (ACC) reflects the prediction accuracy of the model. The value range of ACC is $[0, 1]$. The closer the ACC is to 1, the better the performance of the model is.

$$\text{Accuracy} = 1 - \frac{\sum_{t=1}^T |Y^t - \hat{Y}^t|}{\sum_{t=1}^T Y^t}. \quad (12)$$

- (4) Determination coefficient (R^2) represents the quality of model fitting. The value range is $[0, 1]$. The closer the R^2 is to 1, the better the model fits the data.

$$R^2(Y, \hat{Y}) = 1 - \frac{\sum_{t=1}^T (Y^t - \hat{Y}^t)^2}{\sum_{t=1}^T (\bar{Y}^t - \hat{Y}^t)^2}. \quad (13)$$

- (5) Explained variance score (EVS) is the variance score of the model. The value range is $[0, 1]$. The closer the EVS is to 1, the better the independent variable can explain the variance change of the dependent variable.

TABLE 1: Dataset.

Dataset	Milan telephone service provider
City	Milan
Time span	2013/11/01–2014/01/01
Time interval	10 minutes
Grid size	(100,100)

$$\text{EVS} = 1 - \frac{\sum_{t=1}^T \text{Var}\{Y^t - \hat{Y}^t\}}{\sum_{t=1}^T \text{Var}\{Y^t\}}, \quad (14)$$

where Y^t denotes the actual value of traffic data at the time t and \hat{Y}^t denotes the predicted value of traffic data at the time t . \bar{Y}^t denotes the mean value of traffic data, and T is the number of samples.

4.3. Experimental Parameters. In this experiment, we use a deep learning server to configure the experimental environment, in which the production type of CPU is AMD Ryzen 52600, the production type of GPU is Nvidia GT745 M, the size of Memory is 16 GB. In addition, TensorFlow is used to build the network framework and *Python* is used as the programming environment. Table 2 lists the detailed environment configuration parameters.

Further, we need to determine the model training parameters. In this experiment, Adam is chosen as the optimizer, the learning rate is set to 0.001, and the epoch for model training is 3000. As for the selection of the sliding window length and the number of hidden units, theoretically, on the one hand, the larger the sliding window length is, the larger the perception range will be, and the more features will be predicted, which may cause some interference to the accuracy of prediction. On the other hand, when the number of hidden units increases to a certain extent, the complexity and difficulty of model calculation will also increase, and the accuracy of prediction will also decrease.

Considering that the sliding window length L and the number of hidden units H have a significant impact on the timeliness and accuracy of the traffic prediction, we compared ACC and R^2 under different L and H and obtained the optimal sliding window length and the number of hidden units under the current configuration.

Specifically, the optional range of sliding window length L is set to [4,8,12,16], and by comparing the prediction performance under different L conditions in Figure 7, we obtain the optimal sliding window length, which is 8. That is, we use 8 historical network traffic data ($X_{t-7}, X_{t-6}, X_{t-5}, X_{t-4}, X_{t-3}, X_{t-2}, X_{t-1}, X_t$) to predict future traffic. Similarly, the optional range of the number of hidden units H is set to [32,64,100,128], and by comparing the prediction performance under different H conditions in Figure 8, we obtain the optimal number of hidden units, which is 100.

In conclusion, when the sliding window length is set to 8 and the number of hidden units is set to 100, the prediction result is optimal. Therefore, the model training parameters containing the above results are listed in detail in Table 3.

4.4. Result Analysis

4.4.1. Comparison Results between AGG Model with Other Baseline Models. To verify the performance of AGG model, 80% of traffic data are selected as the training dataset, and 20% of traffic data are selected as the verification dataset. The comparison indicators are described in Section 4.2. In addition, five baseline models are selected including model-driven methods and data-driven methods to compare with the model proposed in this paper. The comparison results are listed in Table 4; because the sampling interval of the traffic data is 10 minutes, we use 10 minutes (one point) and 20 minutes (two points) to carry out single-step prediction and multistep prediction, respectively.

- (1) Historical average model (HA), which models network traffic as a periodic process to predict the time series
- (2) An autoregressive moving composite average model (ARIMA), which is used to fit the time series into a parameter model for completing the network traffic prediction
- (3) Support vector machine model (SVR), which adopts the machine learning algorithm and uses historical data to fit the relationship between input and output and then predicts future network traffic data
- (4) Gated recurrent unit (GRU), which is an efficient solution to the gradients vanishing issue after a long sequence of inputs
- (5) GCN-GRU, which is a combination model combining a graph convolution neural network (GCN) and a gate control recursive unit (GRU)

Table 4 shows that the experimental indicators of the AGG model proposed in this paper are significantly better than those of other baseline models. To be specific, we have the following:

- (1) At the 10 min prediction span, the AGG model proposed in this paper has optimal values in RMSE, MAE, ACC, R^2 , and EVS. For example, the RMSE of the AGG model is 3.7% lower than that of the GCN-GRU model, 4.2% lower than that of the GRU model, 5.5% lower than that of the SVR model, 6.3% lower than that of the ARIMA model, and 14.7% lower than that of the HA model. The ACC of the AGG model is 1.5% higher than that of the GCN-GRU model, 2% higher than that of the GRU model, 2.3% higher than that of the SVR model, 15.9% higher than that of the ARIMA model, and 6.9% higher than that of the HA model. The AGG model proposed in this paper has optimal values in RMSE, MAE, ACC, R^2 , and EVS. It can be further seen that both AGG and GRU are superior to model-driven traffic prediction methods.
- (2) At the 20 min prediction span, the AGG model proposed in this paper still has optimal values in RMSE, MAE, ACC, R^2 , and EVS. For example, the RMSE of the AGG model is 1.6% lower than that of

9901	9902	...	9999	10000
9801	9899	9900
...
101	102	200
1	2	3	...	100

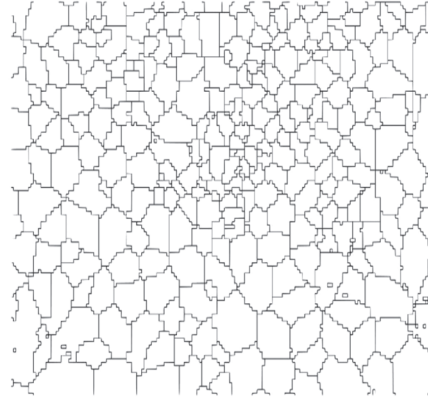


FIGURE 5: Grid and map of the area.

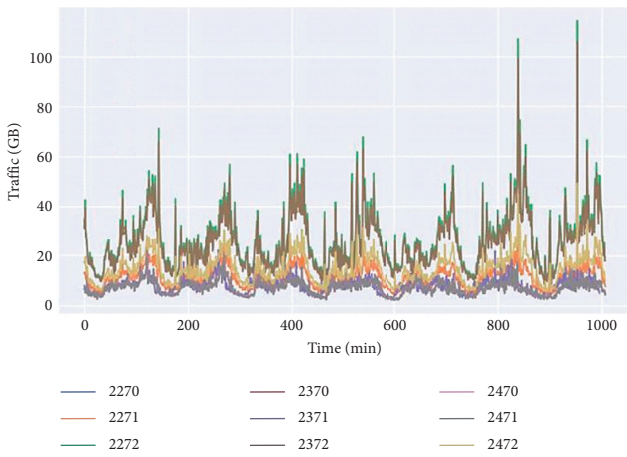


FIGURE 6: Network traffic trend of the nine regions within a week.

TABLE 2: Environment configuration parameters.

Environment component	Parameter
CPU production type	AMD Ryzen 52600
GPU production type	Nvidia GT745 M
Memory size	16 GB
TensorFlow version	1.4
Python version	3.7

the GCN-GRU model, 1.7% lower than that of the GRU model, 1.9% lower than that of the SVR model, 2.5% lower than that of the ARIMA model, and 7.4% lower than that of the HA model. The prediction accuracy of the AGG model is 0.7% higher than that of the GCN-GRU model, 1.8% higher than that of the GRU model, 2.5% higher than that of the SVR model, 12.2% higher than that of the ARIMA model, and 3.5% higher than that of the HA model.

- (3) It can be further concluded from the prediction results that, in horizontal comparison, the data-driven prediction methods, whether SVR or GRU, are better than other model-driven methods. This result is due to the poor fitting ability of HA and ARIMA for this long series of unstable data, while

the neural network models fit the nonlinear data much better. In longitudinal comparison, the performance indicators of the AGG model proposed in this paper decrease with the increase of prediction time, but the decline trend is relatively stable, and it still has long-term prediction ability.

4.4.2. *Influence of Spatial-Temporal Correlation and Attention Mechanism on Prediction Performance.* In order to further explore the influence of spatial-temporal correlation and attention mechanism on prediction performance, two experimental indicators, RMSE and ACC, are used to compare AGG model with other baseline models at the 10 min prediction scale, and the comparison results are shown in Figures 7 and 8, respectively.

Figure 9 shows the comparison results of RMSE between AGG model and other baseline models. These baseline models include model-driven traffic prediction methods HA and ARIMA, and data-driven traffic prediction methods SVR, GRU, and GCN-GRU. Specifically, RMSE of model-driven traffic prediction method are 6.1774 (HA) and 5.6241 (ARIMA) respectively, and RMSE of data-driven traffic prediction method are 5.5817 (SVR), 5.4932 (GRU), 5.4761 (GCN-GRU), and 5.2721 (AGG), respectively. Therefore, RMSE on the whole presents a downward trend, and the AGG model proposed in this paper has the smallest RMSE, which means that the model of spatial-temporal correlation and the introduction of an attention mechanism are fundamental to reduce the RMSE of network traffic prediction results.

Figure 10 shows the comparison results of ACC between AGG model and other baseline models. These baseline models are consistent with Figure 9. Specifically, ACC of model-driven traffic prediction method are 0.6785 (HA) and 0.6264 (ARIMA), respectively, and ACC of data-driven traffic prediction method are 0.7095 (SVR), 0.7114 (GRU), 0.7150 (GCN-GRU), and 0.7256 (AGG), respectively. Therefore, ACC on the whole presents an upward trend, and the AGG model proposed in this paper has the largest RMSE, which means that the model of spatial-temporal correlation and the introduction of an attention mechanism are significant to improve the ACC of network traffic prediction results.

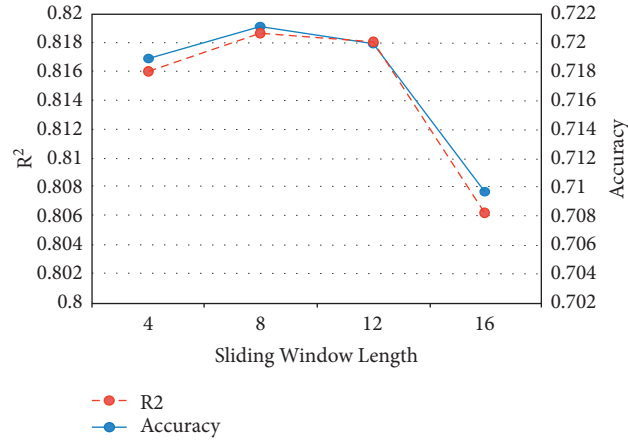


FIGURE 7: Performance comparison under different sliding window lengths.

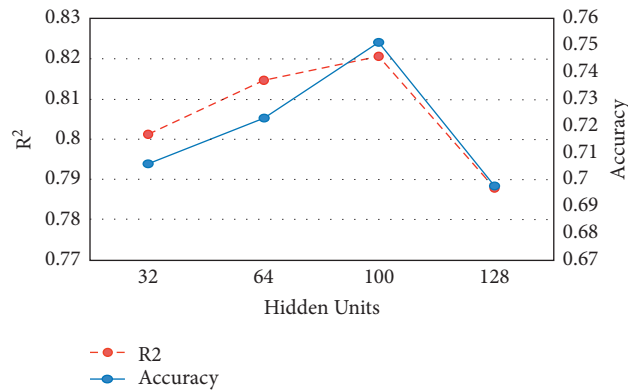


FIGURE 8: Performance comparison under different number of hidden units.

TABLE 3: Model training parameters.

Model component	Parameter
Batch size	32
Learning rate	0.001
Training epoch	3000
Sliding window length	8
Hidden units	100

4.4.3. Analysis of Visual Results of Traffic Prediction. In order to more intuitively see the prediction results of the proposed AGG model, Figures 11 and 12, respectively, show the traffic trend comparison diagram between the prediction value and the true value of AGG model in 10 min and 20 min prediction spans of area 2270. In the experiment, the sliding window length is set to 8 and the number of hidden units is set to 100, which has been proved in Section 4.3 that these parameters are optimal.

It can be seen from Figures 11 and 12 that the AGG model proposed in this paper has good prediction performance, but it has the following two flaws. On the one hand, the prediction result of network traffic at the peak is poor. The main reason is that the GCN model defines a smoothing

filter in the Fourier domain and captures the spatial characteristics by continuously moving the filter and signal for winding operation. This process leads to smoother prediction of the mutation region. On the other hand, there is a certain error between the true network traffic data and the prediction results. The possible reason is that when there is no communication at a certain time in the region, the value of network traffic may be zero, or the value of network traffic may be very small, and a small difference may cause a large relative error. Further, by comparing Figures 11 and 12, we can also get that, with the increase in the prediction time scale, the fitting level between the prediction value and the actual value also decreases, indicating that the small prediction scale always has a better prediction effect.

TABLE 4: Comparison results between AGG model with other baseline models.

T (min)	Metric	HA	ARIMA	SVR	GRU	GCN-GRU	AGG
10	RMSE	6.1774	5.6241	5.5817	5.4932	5.4761	5.2721
	MAE	3.2447	3.9729	2.8332	2.7525	2.7009	2.5788
	ACC	0.6785	0.6264	0.7095	0.7114	0.7150	0.7256
	R^2	0.7613	0.0356	0.8051	0.8078	0.8124	0.8261
	EVS	0.7613	0.0008	0.8065	0.8086	0.8125	0.8265
20	RMSE	6.1774	5.8655	5.8321	5.8211	5.8189	5.7209
	MAE	3.2447	3.9767	2.9926	2.9836	2.9732	2.8362
	ACC	0.6785	0.6262	0.6856	0.6898	0.6973	0.7024
	R^2	0.7613	0.0356	0.7749	0.7853	0.7888	0.7959
	EVS	0.7613	0.0009	0.7896	0.7914	0.7931	0.7969

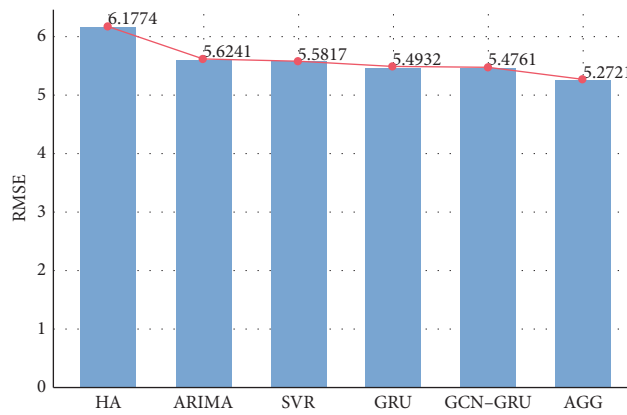


FIGURE 9: Comparison results of RMSE between AGG model and other baseline models.

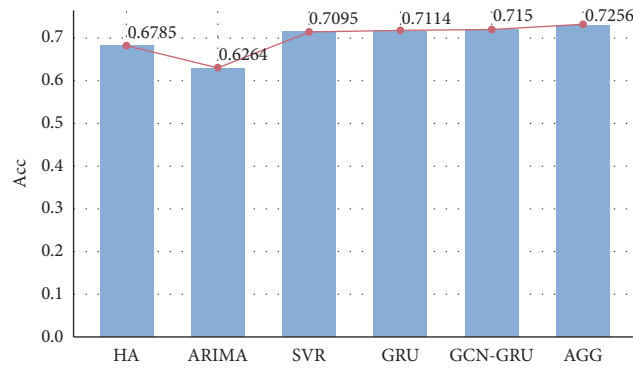


FIGURE 10: Comparison results of ACC between AGG model and other baseline models.

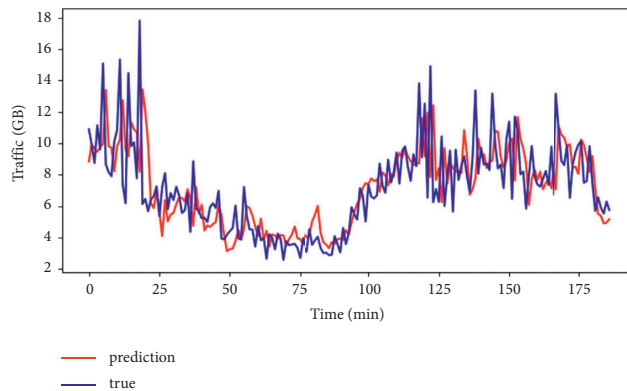


FIGURE 11: Visual results of traffic prediction by AGG model in 10 min span.

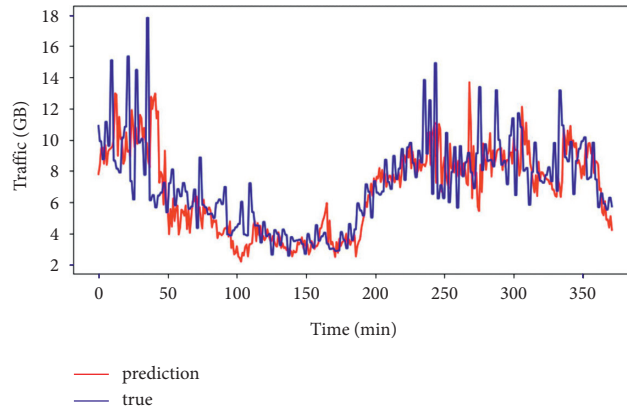


FIGURE 12: Visual results of traffic prediction by AGG model in 20 min span.

5. Conclusion

In this paper, we propose a network traffic prediction method combining GCN, GRU, and attention mechanism. In this method, GCN is used to capture the network topology to obtain the spatial features of network traffic. GRU model is used to capture the dynamic changes of traffic on nodes, so as to obtain the time features of network traffic. Furthermore, the attention mechanism is used to weight the historical traffic data to dynamically adjust the importance of network traffic information at each sampling time. By using the actual network traffic dataset to carry out the experiment and comparing it with the baseline models such as HA, ARIMA, SVR, GRU, and GCN-GRU, it can be concluded that the AGG model proposed in this paper achieves the best prediction effect under different performance indicators.

Data Availability

This paper selects an open network traffic dataset, the download address is <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EGZHFV>, and the traffic collection time is from 00:00 on November 1, 2013, to 00:00 on January 1, 2014.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 61801073, 61722105, and 61931004.

References

- [1] Cisco: Cisco annual internet report (2018–2023) whitepaper [R/OL]. Cisco (2020-03-09) [2020-05-15]. <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internetreport/white-paper-c11-741490.html>.
- [2] M. Laner, P. Svoboda, and M. Rupp, "Parsimonious fitting of long-range dependent network traffic using ARMA models," *IEEE Communications Letters*, vol. 17, no. 12, pp. 2368–2371, 2013.
- [3] J. Guo, Y. Peng, X. Peng, and Q. Chen, "Traffic forecasting for mobile networks with multiplicative seasonal ARIMA models," in *Proceedings of the Electronic Measurement & Instruments, 2009. ICEMI '09. 9th International Conference on*, August 2009.
- [4] H.-W. Kim, J.-H. Lee, Y.-H. Choi, Y.-U. Chung, and H. Lee, "Dynamic bandwidth provisioning using ARIMA-based traffic forecasting for Mobile WiMAX," *Computer Communications*, vol. 34, no. 1, pp. 99–106, 2011.
- [5] A. Alheraish, S. Alshebeili, and T. Alamri, "Regression video traffic models in broadband networks," *Journal of King Saud University - Engineering Sciences*, vol. 18, no. 1, pp. 19–55, 2005.
- [6] Y. Shu, M. Yu, j. Liu, and O. W. W. Yang, "Wireless traffic modeling and prediction using seasonal arima models," *IEICE - Transactions on Communications*, vol. E88B, no. 10, pp. 1675–1679, 2003.
- [7] P. Bermolen and D. Rossi, "Support vector regression for link load prediction," *Computer Networks*, vol. 53, no. 2, pp. 191–201, 2009.
- [8] G. Kremer, P. Owezarski, and P. Berthou, *Predictive Estimation of Wireless Link Performance from Medium Physical Parameters Using Support Vector Regression and K-Nearest Neighbors*, Springer, Berlin, Heidelberg, 2014.
- [9] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, "A hybrid deep learning based traffic flow prediction method and its understanding," *Transportation Research Part C: Emerging Technologies*, vol. 90, no. 9, pp. 166–180, 2018.
- [10] A. Lazaris and V. K. Prasanna, "An LSTM framework for modeling network traffic," in *Proceedings of the 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE, Washington DC, USA, April 2019.
- [11] A. Azzouni and G. Pujolle, "A long short-term memory recurrent neural network framework for network traffic matrix prediction," arXiv preprint arXiv:1705.2017.
- [12] C. Zhang, H. Zhang, D. Yuan, and M. Zhang, "Citywide cellular traffic prediction based on densely connected convolutional neural networks," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1656–1659, 2018.
- [13] L. Zhao, Y. Song, C. Zhang, and Y. Liu, "T-GCN: a temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 99, pp. 1–11, 2019.

- [14] K. He, Y. Huang, X. Chen, and Z. Zhou, "Graph attention spatial-temporal network for deep learning based mobile traffic prediction," in *Proceedings of the GLOBECOM 2019 - 2019 IEEE Global Communications Conference*, December 2019.
- [15] L. Yang, X. Gu, and H. Shi, "A novel satellite network traffic prediction method based on GCN-GRU," in *Proceedings of the 2020 International Conference on Wireless Communications and Signal Processing (WCSP)*, Xi'an, China, October 2020.
- [16] J. Zhu, Y. Song, L. Zhao, and H. Li, "A3T-GCN: attention temporal graph convolutional network for traffic forecasting," arXiv:2006.11583, 2020.
- [17] H. Zhang, L. Chen, J. Cao, and X. Zhang, "A combined traffic flow forecasting model based on graph convolutional network and attention mechanism," *International Journal of Modern Physics C*, 2021.

Research Article

A Hybrid Association Rule-Based Method to Detect and Classify Botnets

Yuanyuan Huang ¹, Lu Jiazhong ¹, Haozhe Tang ¹ and Xiaolei Liu ²

¹School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, Sichuan, China

²Institute of Computer Application, China Academy of Engineering Physics, Mianyang, Sichuan 621900, China

Correspondence should be addressed to Xiaolei Liu; liuxiaolei@caep.cn

Received 19 May 2021; Revised 4 August 2021; Accepted 25 August 2021; Published 17 September 2021

Academic Editor: Weiwei Liu

Copyright © 2021 Yuanyuan Huang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Nowadays, botnet has become a threat in the area of cybersecurity, and, worse still, it is difficult to be detected in complex network environments. Thus, traffic analysis is adopted to detect the botnet since this kind of method is practical and effective; however, the false rate is very high. The reason is that normal traffic and botnet traffic are quite close to the border, making it so difficult to be recognized. In this paper, we propose an algorithm based on a hybrid association rule to detect and classify the botnets, which can calculate botnets' boundary traffic features and receive effects in the identification between normal and botnet traffic ideally. First, after collecting the data of different botnets in a laboratory, we analyze botnets traffic features by processing a data mining on it. The suspicious botnet traffic is filtered through DNS protocol, black and white list, and real-time feature filtering methods. Second, we analyze the correlation between domain names and IP addresses. Combining with the advantages of the existing time-based detection methods, we do a global correlation analysis on the characteristics of botnets, to judge whether the detection objects can be botnets according to these indicators. Then, we calculate these parameters, including the support, trust, and membership functions for association rules, to determine which type of botnet it belongs to. Finally, we process the test by using the public dataset and it turns out that the accuracy of our algorithm is higher.

1. Introduction

Botnet is a group of centrally controlled bots on the Internet, and these computers using the botnet are called controlled hosts, which are often utilized by hackers to launch a large-scale cyberattack. These computers contain spams port scans, phishing sites, etc. The botnet host can also control the information stored in those computers, such as passwords of bank account and social accounts. In the meantime, hackers can also get the function of "access" of their computers easily. No matter it is the safe operation of the network or the users' data security protection, the botnets are perilous risks. However, current technology cannot recognize those botnets easily for they are usually controlled by hackers long distantly. In other words, users are often unaware of these hosts.

Nowadays, the main botnet detection algorithms are to detect network traffic. The existing detection methods have

some shortcomings, however. For instance, if we only rely on bots' similarity detection method, the result is prone to get false. When it is difficult to determine the number of clusters by using the clustering algorithm, we need to establish a blacklist to complete the test. However, the blacklist depends on the bot's malicious attacks, and the efficiency of detection will be quite low.

According to different classification criteria, there are several classification methods of botnet detection: host-based detection methods, network-traffic-based detection method, and real-time detection methods. Host-based detection methods detect botnets by analyzing information logs and acting on the host. Because botnets will bring a series of changes while running, such as changing the registry, skipping the firewall, establishing a network connection, bypassing intrusion detection, and turning off antivirus software[1]. System changes caused by bots and

legal programs are very different; thus, some research detected botnets by analyzing the host information [2–5].

Host-based detection method detects at a faster speed but with a lower cost. However, to apply host-based detection, the method needs to install specific software on each host, indicating its poor expansibility and adaptability. In addition, because of the various information in hosts, the formats of different operating systems are not the same information sources, which make this method difficult to be adopted. Therefore, the detection methods based on network traffic become the mainstream. Moreover, there is a method combining host and network traffic, and we can get the details from the literature [6, 7].

Methods based on network traffic detection can be divided into two types: active detection and passive detection. Active detection sends probe packets to the Internet to detect the existence of bots, while passive detection collects network traffic passively, analyzing and processing network traffic to detect botnets after that.

Active detection method has higher detection efficiency, it can detect whether there exists a botnet swiftly. However, the active detection method has some obvious shortcomings: the probe packets sent with the help of this method will add additional traffic to the network. It means that attackers can easily find them out and then avoid being detected.

In most of the time, passive detection technology can acquire network traffic from the measured network core, switch firstly, then analyze and process the collected traffic, and finally detect botnets. Passive detection technology will not generate extra traffic, and so attackers will not find it easily.

The detection method based on real-time reduces the detection time to a few seconds [8] without affecting the detection accuracy. Because of Botnets' long delay in the HTTP response, it can be used as a result of request relaying through the botnet proxy. This process usually takes extra time, and the nodes associated with the botnet agent have relatively limited calculating capability and network bandwidth. The real-time detection method may produce a relatively high false alarm rate because it may misclassify a legitimate web server as a malicious domain name.

2. Related Work

In our previous research [9], we have proposed an effective botnet detection method based on fuzzy association rules (FARR). This method can calculate the features of botnet traffic accurately, which can be used to recognize the normal traffic and botnet traffic, while the false alarm rate is relatively high.

Perdisciet et al [10] suggested that we should adopt the real-time tracking method (including queries) of DNS traffic, collect DNS responses by inserting monitors at some key positions in the ISP network, and analyze the traffic to facilitate searching for the coverage area of the botnet. The C4.5 decision tree classifier is used to classify the domain names. Different from the active detection method, the

advantage of this method is that it does not create extra load on network resources to form active DNS queries and requests. It also makes it impossible for botnet attackers to detect these traces. However, such systems have a high false rate relatively. In addition, the detection delay of this technique is longer than any other task.

Tyagi and Aghila [11] proposed an analysis-based detection technique (ABDT) specifically for detecting botnets using a geographically dispersed set of proxy hosts with FFSN. HT Wang et al. [12] proposed a method for identifying botnets in real time by using a Local Spatial Geolocation Detection (LSGD) system, while also using Autonomous System Numbers (ASNs) to enhance localized geographic features. Huang et al. [13] proposed a Spatial Snapshot Fast-Flux Detection (SSFD) system based on two new spatial measuring methods: spatial distribution estimation and spatial service relationship evaluation. This system can capture the geographic location of the host, and the IP addresses from the response to the DNS response are mapping to the geographic coordinate system to detect the fast-flux botnets in real time and mitigate the harm caused by it.

Kang et al. [14] proposed a method of passive P2P monitor (PPM) which can identify the infected host's firewall or NAT. This method is derived from the authors of the study after the Storm Botnet, that is, the probability of establishing the coverage model (probability-based coverage model). The authors also utilized a verification tool (Firewall Checker, FWC) to verify the result of the identification. Research results suggest that 40% of the infected hosts are being used after a firewall or a NAT.

Saad et al. [15, 16] adopted the method of feature extraction of network traffic to detect P2P botnets, and this paper presents a dozen of feature values of network traffic, including the length of load average packet, the number of packets switching, and packet averaging intervals. Then they used machine learning methods to build a classifier to detect P2P botnets.

Wang et al. [17] proposed a fuzzy recognition algorithm to detect botnets. The paper points out that they regularly have DNS traffic and TCP traffic, and there are three steps to detect botnets: first, reducing the traffic to improve the detection efficiency; second, distributing the data packet whose feature is regarding the total number of DNS queries and the number of failures of DNS queries as the feature of DNS traffic. Meanwhile, we use TCP queries and response time distribution, the total number of TCP queries, and TCP data stream size distribution as TCP traffic's feature. Finally, we utilize the fuzzy recognition algorithm to detect domain names and IP addresses associated with botnets, thereby detecting botnets.

To solve the above problems, we propose a hybrid association rule algorithm to detect and classify the botnets. It includes global associations and fuzzy associations, and it also adds the detection of fast-flux botnets. Global associations can detect whether the data are a botnet, and fuzzy associations use global associations to determine what type of botnet it is. The results show that we can detect botnets quite well, to classify the botnets well.

3. Our Approach

According to the characteristics of traffic in a high-speed network environment, we propose a suspicious traffic filtering method based on real-time characteristics to reduce the total traffic that the system needs to process, the resources' consumptions, and improve the system performance. Taking the limitations of time-based detection methods currently into consideration, a hybrid association botnets detection method is derived according to the global association features extracted from the idea of bipartite graphs and combining local time features with fuzzy recognition. It includes global associations and fuzzy associations, which is shown clearly in Figure 1.

In global association, we analyze the global association between domain names and IP addresses, and we apply XGBoost machine learning algorithm with high detection speed and accuracy to botnet's detection to improve the accuracy of detection further. Consequently, it also enriches the botnet's dimension of the feature vector, breaks the limitations of the current research methods, improves detection efficiency and accuracy, and reduces detection false alarms and the rate of false alarms.

In fuzzy recognition, we have to divide the extracted features in a strict way. More specifically, different levels represent different degrees, the closer the botnet, the closer the level of optimization features to the botnet. According to the fuzzy algorithm principle of the maximum degree of membership, we can determine the attributes of the dataset. We also propose botnet association rules based on support and confidence formulae, and they can be used for mining association rules between botnets' features, which help us to determine the type of botnets and recognize normal and abnormal data.

3.1. Data Type. We set up botnet environment and collect data through public datasets. 36 normal datasets, 33 botnets datasets, 3 public botnets datasets [18], and 13 public botnets datasets [19] are collected together and shown in Table 1. We collected data and published three datasets for doing traffic analysis, and these abnormal datasets contain IRC, HTTP, P2P, fast-flux, and other botnets.

Besides, we utilize the real blacklist to access the traffic generated and the ISOT botnets' data collection and recombination and then utilize the TCPReplay tool to highly simulate the high-speed environment and replay the combined data stream packets. The high-speed network environment used in the test utilizes TCPReplay tool to simulate a 1 Gbps network and a 10 Gbps network, respectively. Table 2 shows the sources for collecting blacklists.

At this stage, the malicious domain names were collected for up to 48 hours. To increase the diversity of data, the top 500 popular domain names of Alexa [20] were selected for collection, and a 2.32 GB data stream package was chosen.

3.2. Traffic Filtering. The real-time suspicious traffic filtering method combines the advantages of black/white list and general real-time feature filtering of botnets to enhance the

real-time and relative accuracy of filtering. In a complex network environment, the real-time detection of suspicious botnet domain names can further reduce and clean up complex DNS traffic, to provide an effective DNS data stream for the subsequent accurate detection, improve the system's speed of filtering DNS traffic, and reduce the overhead of system resources. Table 1 indicates the processing flow of the suspicious traffic filtering method based on real-time characteristics.

The real-time filtering methods for DNS traffic are here mentioned as follows:

- (1) *Protocol Filtering.* The DNS parsing service uses port 53 for data transmission. Therefore, the first step is to use port 53 and the DNS packet header to filter DNS traffic.
- (2) *Black and White Lists' Filtering.* The DNS traffic generated by most users on the Internet is harmless. A whitelist-based filtering method can filter a large amount of benign DNS access data, to reduce the data that the algorithm will use quickly and in real-time. Speaking of a specific fast-flux botnet real-time detection method that cannot distinguish the defects of the CDN network and the fast-flux network, we use the first 100,000 domain names of Alexa, which can filter most CDN networks. The blacklist can directly filter malicious domain names, then alert the user and store it in the blacklist database, which provides technical support for mixed association botnet detection methods.
- (3) *The Real-Time Feature Filtering of Botnets.* In real-time detection of botnets, feature vectors relatively are used to improve the real-time performance of the detection algorithm. However, due to the similarity between the CDN network and the fast-flux botnet, the real-time detection method has a high false positive rate and false negative rate. This article summarizes some general characteristics of real-time detection based botnets, relaxes filtering rules, eliminates false alarms, and filters suspicious fast-flux traffic, to provide accurate and effective data for the following algorithms, which can improve the detection performance and effectiveness.

3.3. Feature Extraction. We divide the crawling traffic into UDP and TCP flow by following UDP and TCP protocols so that we can count and analyze each flow and packet of datasets. A large number of bots will send a control message to the controlled host. Therefore, when the controlled host accepts messages, it will send it as a response to the bots, where there will be a lot of problems of traffic functions, for instance, a packet being sent successfully, packet transmission time intervals, large amounts of data emanating from the same port, but not containing specific ports.

The method proposed by Wang et al. [17] is inactive botnets, and it changes DNS intervals by the impact of bots. Based on this work, we propose a new method to analyze the TCP protocol of PSH and UDP protocol by utilizing the

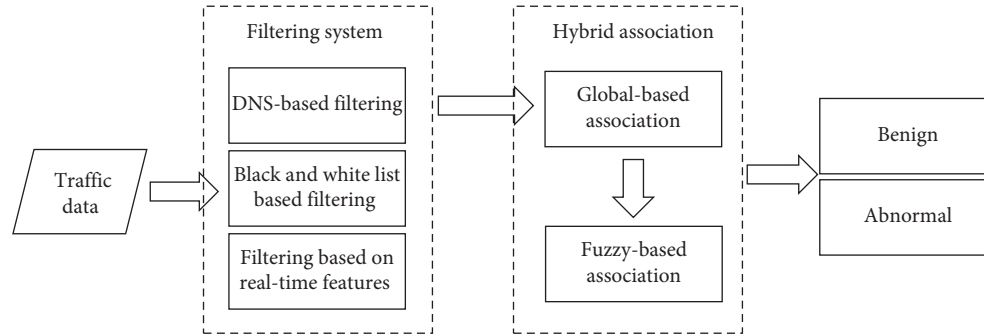


FIGURE 1: Botnet detection and classification process.

TABLE 1: Botnet type, number, and name.

Type	Amount	Botnet name
Normal	33	Normal (ISCX + ISOT)
IRC botnet	24	Neris, Rbot, Menti, Murlo, Tbot, IRC ISCX
HTTP botnets	7	Virut, Sogou
P2P botnets	12	NSIS.ay, SMTP Spam, Zeus (C & C), UDP Storm, Zeus, Zero access, Weasel
PS botnets	3	Zeus
Fast-flux botnets	3	Waledac

TABLE 2: Blacklist.

	Blacklist	Source
1	DNS Blackhole	http://www.malwaredomains.com
2	Spam	http://untroubled.org/spam
3	Phish	http://www.phishtank.com
4	Zeus malicious domain names	http://www.malwaredomainlist.com/forums/index.php
5	ZEUS tracker	https://zeustracker.abuse.ch/blocklist.php
6	Malicious domain names	http://www.abuse.ch/
7	Long-term malicious domain names	http://www.malwaredomains.com/wordpress/?p=1282

DNS response intervals. If there is no active botnet, there is no data needed to transmit, and PSH will be 0. The proportion of PSH in the dataset has changed, and DNS response will have a fixed interval. This will also affect the proportion of the TCP data's, and the source IP ratio will change greatly. When botnet's network node changes or controlled host strengthens defense, it will affect the success rate of data transmission. Some botnets transmit data through the C & C server, which relates to the existence of a specific port. Among the fixed TCP ports, the fixed UDP port, and interval DNS request, one is Boolean attribute, while another is a quantitative attributes.

In the TCP process, we need to analyze the following features, and Table 3 shows the related statistics.

We also select real-time function to further refine DNS traffic after the filtering of black/white lists. Paper [21] mentioned if any DNS A records TTL = 0, that domain will be marked as suspicious. If the TTL is not 0, we use the real-time characteristics in Table 4 to classify the domain into suspicious domain names or benign domain names. In each DNS response, both the A record and the NS record have a TTL field, which is used to specify the response retention time, or it means the effective intervals of the DNS cache.

Although the RFC suggests calculating the minimum TTL in days, most legitimate high-availability websites use TTL values between 600 and 3600 seconds.

It is worth noting that in some fast-flux botnets, to change the IP address and IP of NS servers quickly, the attacker usually uses a TTL value of less than 300 seconds so that bots can connect to C & C hosts in time. In addition, to achieve the better load balancing and higher fault tolerance ability, the existing content distribution and Round-Robin DNS (RRDNS) networks usually have a smaller TTL value. Table 5 shows the TTL values of the types of network A's records.

Most benign Fully Qualified Domain Names (FQDNs) are mapped to even closer hosts and are part of the same ASN. Some fast-flux zombie hosts are geographically dispersed on the Internet randomly. Their characteristics belong to different autonomous systems. The A and NS records for domain names also occupy some more countries. Therefore, all IP addresses of a domain have the same ASN and country/region, which means that the domain is legal; otherwise, it may be suspicious. Table 6 lists and shows the number of ASN distributions and country distributions of benign and fast-flux domain names.

TABLE 3: Botnet features.

TCP protocol	PSH = 1 proportion the dataset TCP packet incoming and outgoing ratio ICMP success rate of sending Containing a specific TCP port, such as 6665,6667,8000,9000 Source IP proportion
UDP protocol	DNS request interval 90–110 s The same UDP port proportion of all ports The highest proportion of fixed UDP port

TABLE 4: Selection of global correlation features.

Category	Description
TTL value	A recorded survival time NS recorded survival time
The diversity of ASN	Diversity of ASN (autonomous domain number) of IP address in a record Diversity of ASN (autonomous domain number) of IP addresses in NS records
Number of IP addresses	Number of IP addresses in the a record Number of name server IP addresses in NS records

TABLE 5: TTL for fast-flux and high-availability networks.

Fast-flux botnets		High-performance benign network	
Domain name	TTL (s)	Domain name	TTL (s)
jaaphram.com	60	yahoo.com	1574
p-alpha.ooo.al	60	google.com	52
prtscrinsertcn.net	60	youtube.com	129
Entryrxshop.com	300	baidu.com	455
towardplian.com	120	163.com	444
gty5.ru	542	microsoft.com	3600
mp3for-you.com	60	huya.com	600

TABLE 6: ASN and country distribution number of fast-flux and normal domain names.

Fast-flux botnets			High-performance benign network		
Domain name	ASN	Country	Domain name	ASN	Country
leddamp.com	98	54	taobao.com	1	1
envoyee.com	112	31	renren.com	1	1
spampro.info	55	23	qq.com	1	1
leolati.com	102	30	baidu.com	2	1

In the fast-flux botnet, the IP address corresponding to the domain name is constantly changing. The global association mapping between the extracted domain name and the IP address is shown in Figure 2. The central points in the figure represent the domain name nodes, and the divergent nodes are represented as IP nodes. The global feature extraction is shown in Table 7.

3.4. *Clustering Features and Dividing the Boundaries.* Effective botnet’s feature discretization is the key to the mining association rules; it is completely based on the method of part K to support the existence of inadequacies, especially when dealing with the difficulty in reflecting the actual distribution result from the high skewness of data

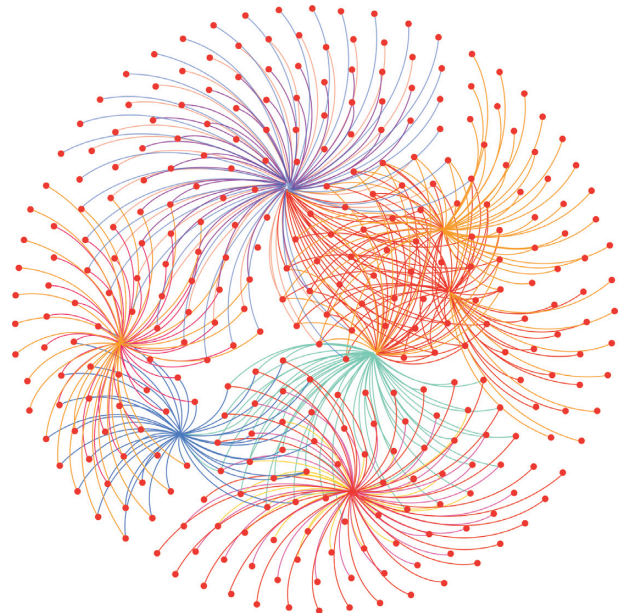


FIGURE 2: Association map of suspicious mapping of malicious domain names based on DNS Map.

effectively. There are excellent demarcation features on the division of the interval. Therefore, by using the FCM algorithm, we divide eight botnets features (including quantitative and Boolean attributes) into number of fuzzy sets; then, such fuzzy sets can convert between a set of elements and nonelements, to achieve softening the feature attributes of demarcation. When dealing with high skewness of data, FCM algorithm can effectively reflect the actual distribution of the data.

To classify the botnet features accurately, we must use various types of botnet datasets. In the FCM clustering of botnet matrix, 50 iterations, we divide it into five categories, and sizes of the center are divided into higher, high, medium, low, and lower, represented by numbers 5, 4, 3, 2, 1,

TABLE 7: Selection of global correlation features.

Category	Description
Number of nodes	Number of IP addresses Number of FQDN
Node degree	Maximum and average degrees of FQDN nodes
Betweenness	Betweenness FQDN node's largest IP node intermediary centrality

respectively. Based on the calculated botnets feature matrix and the center, the level of fuzzy sets can be determined by comparing the size of each of the centers. The largest center of fuzzy sets corresponding to the maximum level and the largest center of the corresponding elements of the matrix rows is the botnet ambiguity on fuzzy set maximum level. We list the feature PSH and ICMP's original datasets and classify those fuzzy sets. According to the different features of botnets classification, we can conclude the membership of each dataset's features, which means the features of botnets are different from each other. In other words, because of the different botnet's control nodes, the way of delivering messages, sending commands, and controlling the controlled host should be different.

We use fuzzy clustering to divide the quantitative features of botnets into five ranges, and we use the same quantitative feature values as the target dataset. In each interval in each class, the maximum and minimum values are taken as a maximum and a minimum range respectively. Thus, the quantitative attributes of Botnets could be divided into five ranges. In these fuzzy intervals, we can better respond to the actual distribution of botnet functions.

After analyzing all botnet datasets, we found that most botnets are at a lower level in PSH function. TCP packet incoming and outgoing ratio stay at two different points, one is at a high level while another is lower. We believe that different botnets have different proportions; IP source distribution is mainly in the higher level, and ICMP success rate is in a lower level, while the same UDP port functions

are still in the lower level. DNS intervals, TCP, and UDP fixed port are distributed evenly.

With the basis of the conclusions acquired before, we divided the definition of this characteristic range dataset into five levels: higher, high, medium, low, and lower. These five levels represent the probability of Botnets as very high, high, medium, low, and very low, respectively. In Boolean features, we use different types of statistical methods to assess each level. Finally, we obtain a more accurate range of Botnet features, as it is shown in Table 8.

4. Association Rules for Botnet Recognition

In most botnets existing between features and feature necessary links, some are very closely linked while some are not, so our goal is to find features linked closely.

4.1. Botnet Fuzzy Association Rules. According to the level of the dataset to be divided, the numerical feature values will be converted into approximate Boolean feature values.

We set the botnet dataset feature dimension as p , class label dimension as q , then

$$\begin{aligned} X &= \{y_1, \dots, y_p\}, \\ Y &= \{y_{p+1}, \dots, y_{p+q}\}. \end{aligned} \quad (1)$$

Botnet datasets and class labels can be expressed as

$$S = \left\{ s_j = (X, Y)_j = (y_1, \dots, y_p; y_{p+1}, \dots, y_{p+q})_j = (y_{j1}, \dots, y_{jp}; y_{j(p+1)}, \dots, y_{j(p+q)}) \mid j = 1, \dots, n \right\}. \quad (2)$$

Fuzzy sets of botnets features and class labels can be expressed as

$$MF = \{A_m^k(y_{jm}) \mid j = 1, \dots, n; m = 1, \dots, p + q; k = 1, \dots, k_{jm}\}. \quad (3)$$

Then,

$$FSup(X) = \sum_{j=1}^n \prod_{m=1}^p t_j(y_m) = \sum_{j=1}^n \prod_{m=1}^p \max_{k=1, \dots, k_{jm}} \{A_m^k(y_{jm})\}. \quad (4)$$

Fuzzy association rule " $X \Rightarrow Y$ " fuzzy support is calculated by

$$FSup = \sum_{j=1}^n \prod_{m=1}^{p+q} t_j(y_m) = \sum_{j=1}^n \prod_{m=1}^{p+q} \max_{k=1, \dots, k_{jm}} \{A_m^k(y_{jm})\}. \quad (5)$$

Fuzzy association rule " $X \Rightarrow Y$ " fuzzy confidence is calculated by

$$Fconf = \frac{FSup(X \cup Y)}{FSup(X)}. \quad (6)$$

TABLE 8: Botnet features of range.

Features	Higher	High	Medium	Low	Lower
PSH = 1	0–4.2%	18.5–32.7%	66.8–84.4%	94.3–100%	44.1–47.1%
TCP packet IN/OUT	122.8–160%	0–22.2%	25.1–58.2%	75.9–97.2%	218–230%
Source IP	18.5–36.4%	70.0–82.1%	0–8%	85.5–89.9%	57.5–59.4%
ICMP rate	0–8.5%	35.7–42.4%	27–25.2%	98.9–99.9%	92.9–97.2%
UDP port	0–9.7%	38.5–52.8%	66.7–71%	84.7–100%	22.3–36.3%

We define the minimum support as equal to 0.06 and minimum confidence equal to 0.2, which is a meaningful association rule. According to the different features of different memberships and formula, we calculate the botnet's association rules. Antecedents $i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_9, i_{10}$ represent 10 botnet features; then, the i_{10} indicates the datasets' types: normal, Botnet, IRC, P2P, HTTP, Fast-Flux, Mix (IRC, P2P, HTTP, and PS), and part of fuzzy association rules. i_9 represents fast-flux botnet TTL < 300. i_{10} represents fast-flux botnet number of ASN distributions > 2. As shown in Table 9, if X is A , then Y is B , $X = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8\}$, $Y = \{i_{11}\}$, A represents the association rules, and B represents the properties of Y .

Selecting meaningful association rules from the table of analysis, we can obtain flag i_6, i_7, i_8 , which are used to determine the flag of botnets. Normal dataset features i_6, i_7, i_8 are all equal to 0, which is a high degree of confidence.

After calculation, we get $FSup(i_7) = 0.72$, $FConf(i_7) = 0.566$, which can be explained in the botnet. The same source IP distribution must be accounted for a large proportion of the whole IP, which explains the bots will always continue to send information to the target host.

Botnet features i_6, i_7, i_8 for IRC botnet also have strong association rules. It can be used to identify IRC Botnets. In other features, we can find IRC Botnets being divided into two categories, the first is the lower the transmission effects are, the lower incoming packets outgoing ratio will be. The source IP distribution is obvious and there are DNS requests frequently, indicating that this type of IRC botnets has breakpoints in network nodes. In other words, controlled hosts add defensive measures to prevent bots' control. The second category is that the botnet has high efficiency of transmitting data. Transmission data packet also increased, indicating that this type of Botnets is very active.

When $i_3 = 3 \wedge i_5 = 3 \wedge i_8 = 1$, $i_9 = \text{http}$ botnet and support = 6.1% and confidence = 50%. The proportion of http botnet explained in the same UDP port and the proportion of port number 0&161 have obvious characteristics. For identifying the http botnet, it has great help.

When $i_1 = 2 \wedge i_2 = 1 \wedge i_5 = 1 \wedge i_6 = 1 \wedge i_8 = 0$, $i_9 = \text{mixed}$ botnet. At this time, support = 11.5% and confidence = 42.8%.

It is noteworthy that when $i_4 = 1$, ICMP has a high success rate. It may also be associated with a strong botnet or a normal dataset.

4.2. Detection of Botnets Based on Hybrid Associations. Hybrid association detection is a more accurate analysis and detection of filtered suspicious network traffic. First, we check whether there exists a botnet in the controlled

TABLE 9: Part meaningful association rules.

Rules	FSup	FConf
$i_6 = 1 \wedge i_7 = 1 \wedge i_8 = 1 \Rightarrow \text{IRC}$	0.091	0.261
$i_6 = 1 \wedge i_7 = 1 \wedge i_8 = 0 \Rightarrow i_{11} = \text{IRC}$	0.091	0.200
$i_1 = 1 \wedge i_2 = 5 \wedge i_6 = 0 \wedge i_7 = 1 \Rightarrow \text{IRC}$	0.091	0.261
$i_1 = 4 \wedge i_2 = 1 \wedge i_3 = 1, i_7 = 1 \Rightarrow \text{IRC}$	0.091	0.261
$i_6 = 0 \Rightarrow \text{IRC}$	0.393	0.684
$i_7 = 1 \Rightarrow \text{IRC}$	0.424	0.736
$i_8 = 1 \Rightarrow \text{IRC}$	0.424	0.736
$i_6 = 0 \wedge i_7 = 0 \wedge i_8 = 0 \Rightarrow \text{Normal}$	0.530	0.97
$i_1 = 2 \wedge i_2 = 5 \wedge i_6 = 0 \Rightarrow \text{P2P}$	0.375	0.50
$i_1 = 2 \wedge i_2 = 1 \wedge i_5 = 1 \wedge i_6 = 1 \wedge i_8 = 0 \Rightarrow i_{11} = \text{Mix}$	0.109	0.42
$i_3 = 3 \wedge i_5 = 3 \wedge i_8 = 1 \Rightarrow i_9 = \text{HTTP}$	0.061	0.50
$i_3 = 5 \Rightarrow i_{11} = \text{botnet}$	0.472	0.566
$i_4 = 1 \Rightarrow \text{botnet}$	0.303	0.606
$i_{11} = 1 \Rightarrow \text{normal}$	0.470	0.939
$i_9 = 1 \wedge i_{10} = 1 \Rightarrow \text{Fast-flux}$	0.236	0.710
$i_9 = 1 \wedge i_{10} = 0 \Rightarrow \text{normal}$	0.391	0.452

network; then, the global feature correlation and local feature mixing methods are utilized to detect the botnet. For the lack of botnet detection, it greatly improves the detection accuracy and efficiency of botnets.

Based on the characteristics of global association and according to the idea of bipartite graph, we observe the global association relationship between the domain name and its mapped IP in a certain period of time. If the IP address hosting the malicious domain name (fast-flux domain name) hosts another unknown domain name, the unknown domain name may also be malicious. Because the fast-flux domain name will map a large number of IP addresses, and attackers can utilize more domain names to organize the fast-flux network. Using the association relationship, we can find the emerging and dying fast-flux domain names. We utilize the DNSMap tool to extract the global mapping relationship between domain names and IP addresses and then calculate the global correlation features, which can enrich the feature vector dimension of the fast-flux botnet.

According to the characteristics of botnets, local characteristics based on time are obtained by parsing DNS data packets and doing statistics on related data. Based on the real-time detection method, we analyze the DNS, and fast-flux domain names can be detected by obtaining 3-4 characteristics. However, the rapid development of existing CDN networks and RRDNS networks has shown the same trend as fast-flux networks' characteristics, so an amount of false alarms is generated during real-time detection. The time-based feature extraction method mainly counts

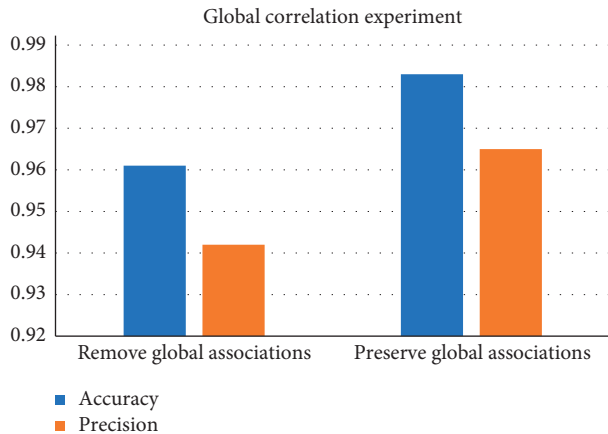


FIGURE 3: Accuracy and precision in the case of ten-fold cross-validation.

information such as the number and growth of IP addresses corresponding to botnet domain names over a period of time, together with the size and average value of TTL values corresponding to domain names. Therefore, although the time-based detection method uses time for detecting, it reduces the false alarm and false alarm rate of the system, which is acceptable in a large high-speed network environment.

During our experiments, we selected 7582 experimental data, which included 77 fast-flux botnet domain names and 7505 benign legal domain names. At the same time, the XGBoost learning algorithm is used for training the data and establishing the optimal classification model.

Then, this experiment verifies the extracted features by doing research on it, mainly to verify the validity of the features based on global correlation. We duplicate the extracted feature dataset into two duplicates, one of which removes the global associated features and only retains the time-based features, while another retains all features. Then we perform ten-fold cross-validation on the feature dataset by utilizing XGBoost machine learning algorithm. Figure 3 shows the accuracy and precision in the case of ten-fold cross-validation.

From Figure 3, we can find that after removing the global correlation features, the detection accuracy and precision have decreased in the case of using ten-fold cross-validation, which have dropped by 2.2% and 2.3%, respectively. At the same time, the number and the rate of false alarms have also decreased. It can be figured out that it is very helpful to improve the detection accuracy and efficiency based on global correlation features.

5. Method Comparison

According to our association rules, the test has already been done with the help of the public data [19]. The accuracy rate can reach 98.2%, and the comparison of various algorithms is shown in Table 10. TPR (true-positive rate) can be understood as how many of all positive classes in the result are predicted to be positive classes in our experiment (correct positive class prediction). FPR (false-positive rate) can be

TABLE 10: Comparison of results.

<i>Public datasets scene 1, IRC botnet</i>		
Method	TNR	FPR
BH	0.5	< 0.0
CA1	0.9	< 0.0
BCLus	0.5	0.4
FARR	0.9	0.1
Hybrid association	0.9	< 0.0
<i>Public datasets scene 2, IRC botnet</i>		
Method	TNR	FPR
BH	0.99	< 0.0
CA1	0.9	< 0.0
BCLus	0.7	0.2
FARR	0.9	0.1
Hybrid association	0.9	< 0.0
<i>Public datasets scene 6, PS botnet</i>		
Method	TNR	FPR
BH	0.99	< 0.0
CA1	0.9	< 0.0
BCLus	0.8	0.2
FARR	0.7	0.2
Hybrid association	0.8	0.1
<i>Public datasets scene 9, MIX botnet</i>		
Method	TNR	FPR
BH	0.99	< 0.0
CA1	0.9	< 0.0
BCLus	0.6	0.3
FARR	0.8	0.1
Hybrid association	0.8	< 0.0
<i>Fast-flux datasets</i>		
Method	TNR	FPR
BH	0.99	< 0.0
CA1	0.9	< 0.0
BCLus	0.6	0.3
FARR	0.8	0.1
Hybrid association	0.9	< 0.0

understood as how many of all negative classes in the result are predicted to be positive classes in our experiment (wrong positive class predictions). We can figure out that using our algorithm to identify the advantages of botnets receives great effects. Meanwhile, it can also mine in their rules deeply, especially for IRC and fast-flux botnets. The reason of it is that we are looking for functions similar to those of IRC and fast-flux botnets.

6. Conclusion

This paper proposes a new hybrid association method to detect and classify botnets. This method can well solve the boundary and classification problems of botnets and normal data. This new detection algorithm contains four steps. First, the collected traffic is filtered by using a suspicious DNS protocol, black/white lists, and real-time function monitoring, which can greatly reduce the detection overhead and improve the detection efficiency. The second is feature extraction for filtered traffic, and it ranges from time-related functions and domain name functions to basic traffic functions. The third is performing global correlation and using machine learning to identify botnets. Finally, we make

the botnets fuzzy associated, determine the association rules by calculating the support and trust degree, and then classify those botnets after that. In botnets' classification, we use various functions between support and confidence degree to filter association rules. We can classify not only IRC and HTTP botnets, but new ones including P2P and fast-flux botnets.

Our next task is to study the double fast-flux botnet and detect it.

Data Availability

The normal and abnormal botnet traffic.pcap data used to support the findings of this study were supplied by Jiazhong Lu under license and so cannot be made freely available. Requests for access to these data should be made to Jiazhong Lu, ljz@cuit.edu.cn.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This paper was supported in part by the National Social Science Fund of China for supporting our project "Computational Communication Strategy for Chinese Technology Image on Twitter," no. 20CXW016, in part by the National Key R&D Program of China (no. 2017YFB0802300), in part by the Key Research and Development Project of Sichuan Province (nos. 20ZDYF0660, 2018GZ0204, 2018RZ0072, and 2021YFYZ0028), in part by the Foundation of Chengdu University of Information Technology under grant J201707 in part by the Project of Chengdu City (no. 2020RK00-00266-ZF), in part by the Director of Computer Application Research Institute Foundation (SJ2020A08), and China Academy of Engineering Physics Innovation and Development Fund Cultivation Project (PY20210160).

References

- [1] T. Micro, *Taxonomy of Botnet Threats*, Trend Micro, Tokyo, Japan, 2006.
- [2] E. Stinson and J. C. Mitchell, "Characterizing bots' remote control behavior," *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer-Verlag, in *Proceedings of the 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA' 07*, pp. 89–108, July 2007.
- [3] L. Liu, S. Chen, G. Yan, and Z. Zhang, "BotTracer: execution-based bot-like malware detection," in *Proceedings of the 11th International Conference Information Security, ISC*, Taipei, China, September 2008.
- [4] L. Liu, S. Chen, G. Yan, and Z. Zhang, "BotTracer: execution-based bot-like Malware detection," in *Proceedings of the 11th International Conference Information Security*, pp. 97–113, Taipei, Taiwan, China, September 2008.
- [5] M. Szymczyk, "Detecting botnets in computer networks using multiagent technology," in *Proceedings of the Fourth International Conference on Dependability of Computer Systems*, pp. 192–201.
- [6] K. Xu, D. Yao, Q. Ma, and A. Crowell, "Detecting infection onset with behavior-based policies," in *Proceedings of the 5th International Conference on Network and System Security (NSS)*, pp. 57–64, ember.
- [7] Y. Zeng, X. Hu, and K. G. Shin, "Detection of botnets using combined host-and network-level information," in *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 291–300.
- [8] C.-H. Hsu, C.-Y. Huang, and K.-T. Chen, "Fast-flux bot detection in real time," in *Proceedings of the International Conference on Recent Advances in Intrusion Detection*, pp. 464–483, Springer-Verlag, Ottawa, Canada, September 2010.
- [9] J. Lu, F. Lv, Q.-H. Liu, M. Zhang, and X. Zhang, "Botnet detection based on fuzzy association rules," in *Proceedings of the 2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 578–584, Beijing China, August 2018.
- [10] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Passive analysis of recursive DNS traces," in *Proceedings of the Computer Security Applications Conference*, pp. 311–320, IEEE Computer Society, Honolulu, HI, USA, December 2009.
- [11] A. K. Tyagi and G. Aghila, "Detection of fast flux network based social bot using analysis based techniques," in *Proceedings of the International Conference on Data Science & Engineering*, pp. 23–26, IEEE, Cochin, India, 18-20 July 2012.
- [12] H. T. Wang, C. H. Mao, K. P. Wu, and H. M. Lee, "Real-time fast-flux identification via localized spatial geolocation detection," in *Proceedings of the Computer Software and Applications Conference*, pp. 244–252, IEEE, Izmir, Turkey, July 2012.
- [13] S. Y. Huang, C. H. Mao, and H. M. Lee, "Fast-flux service network detection based on spatial snapshot mechanism for delay-free detection," in *Proceedings of the 5th International Symposium on ACM symposium on Information, Computer and Communications Security*, pp. 101–111, Beijing, China, January 2010.
- [14] J. Kang, Y. Z. Song, and J. Y. Zhang, "Accurate detection of peer-to-peer botnet using Multi-Stream Fused scheme," *Journal of Networks*, vol. 6, no. 5, pp. 807–814, 2011.
- [15] S. Saad, I. Traore, A. Ghorbani et al., "Detecting P2P botnets through network behavior analysis and machine learning," in *Proceedings of the 2011 9th IEEE Annual International Conference on Privacy, Security and Trust (PST 2011)*, pp. 174–180.
- [16] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer to peer botnet detection based on flow intervals," in *Proceedings of the 27th IFIP TC 11 Information Security and Privacy Conference (SEC 2012)*, pp. 87–102, Heraklion, Greece, June 2012.
- [17] K. C. Wang, C. Y. Huang, S. J. Lin, and Y. D. Lin, "A fuzzy pattern-based filtering algorithm for botnet detection," *Computer Networks*, vol. 55, pp. 3275–3286, 2011.
- [18] Information Security Centre of Excellence, "UNB ISCX botnet DataSet [EB/OL]," 2014, <http://www.unb.ca/research/iscx/dataset/ISCX-botnet-dataset.html>.
- [19] S. García, M. Grill, J. Stiborek, and A. Zunino, "Anempirical comparison of botnet detection methods[J]," *Computers & Security*, vol. 45, pp. 100–123, 2014.
- [20] Amazon company, "Alexa top 500 global sites[EB/OL]," 2020, <https://www.alexa.com/topsites>.
- [21] H. T. Lin, Y. Y. Lin, and J. W. Chiang, "Genetic-based real-time fast-flux service networks detection," *Computer Networks*, vol. 57, no. 2, pp. 501–513, 2013.

Research Article

RT-SAD: Real-Time Sketch-Based Adaptive DDoS Detection for ISP Network

Haibin Shi,^{1,2} Guang Cheng ,^{1,2} Ying Hu,^{1,2} Fuzhou Wang,^{1,2} and Haoxuan Ding^{1,2}

¹School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China

²Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, Nanjing 21189, China

Correspondence should be addressed to Guang Cheng; gcheng@njnet.edu.cn

Received 16 April 2021; Revised 20 June 2021; Accepted 14 July 2021; Published 28 July 2021

Academic Editor: Weiwei Liu

Copyright © 2021 Haibin Shi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the great changes in network scale and network topology, the difficulty of DDoS attack detection increases significantly. Most of the methods proposed in the past rarely considered the real-time, adaptive ability, and other practical issues in the real-world network attack detection environment. In this paper, we proposed a real-time adaptive DDoS attack detection method RT-SAD, based on the response to the external network when attacked. We designed a feature extraction method based on sketch and an adaptive updating algorithm, which makes the method suitable for the high-speed network environment. Experiment results show that our method can detect DDoS attacks using sampled Netflow under high-speed network environment, with good real-time performance, low resource consumption, and high detection accuracy.

1. Introduction

Distributed denial of service (DDoS) attack has been one of the most difficult attacks in the network. DDoS attacks can interrupt the network service temporarily or even make the system break down. DDoS attacks are usually launched by botnet devices. In recent years, the number of IoT devices is increasing rapidly, which are more vulnerable [1] than traditional network equipment. The IoT botnet expands the scales of DDoS attacks significantly. In 2016, DNS service provider Dyn was attacked by massive IoT devices controlled by Mirai Botnet, which directly led to a large area of services unavailable on the east coast of the United States. Another difficulty in defense against DDoS attacks is the rise of reflection amplification attacks. In 2018, GitHub was attacked by a reflection amplification DDoS attack by leveraging the Memcached protocol vulnerability, with the reflection multiple as high as 50,000 times and the peak traffic of 1.35 Tbps.

According to Akamai's annual summary [2] of DDoS attacks in 2020, the number of large-scale DDoS attacks has increased significantly. In the largest DDoS attack event [3],

the attack traffic has reached 1.44 Tbps, and the attack is very complex. It is necessary to combine multiple mitigation methods as soon as possible to block the attack. However, for large-scale DDoS attacks, it is difficult to deploy attack detection and defense devices near the victims for effective defense. A more effective way is to collect traffic and detect DDoS attacks on the backbone network.

In the past decades, researchers have proposed many detection methods for DDoS attacks. Most of the existing methods are based on machine learning or deep learning. These methods need to train the model on a large number of labeled network traffic data in advance to ensure the accuracy of attack detection. However, there are some problems in these methods:

- (a) At present, new attack vectors are constantly being mined. For example, at the end of July 2020, the FBI issued an alert [4] that CoAP, WS-DDARMS, and other protocols may be used to launch DDoS attacks. DDoS attacks based on new attack vectors may have great changes in the statistical characteristics such as packet speed and packet spacing used in traditional

methods, which makes traditional methods less adaptable to different attacks.

- (b) Most of the models need to be trained in advance before they are used for detection. If the network environment changes, the current network traffic may not follow the data distribution of the pre-trained model. At this time, the traditional methods need to retrain the model to maintain high accuracy. However, in the scene of attack detection in the backbone network of operators, it is very difficult to obtain labeled data and retrain the model frequently. In addition, it is also difficult to determine the right time to update the detection model.
- (c) For DDoS attack detection method design and performance evaluation, most of the methods only consider the detection accuracy, false alarm rate, and false alarm rate but do not consider the real-time performance and resource consumption of the method. Although their methods can work in small DDoS attacks simulated by tools such as hping3 [5] and LOIC [6], they did not consider the performance of such methods in the real-world high-speed network environment, like the ISP network.

In order to adapt to various types of DDoS attacks in the high-speed network environment, we propose a real-time adaptive DDoS detection method based on sketch for ISP network. The method implements dynamic adjustments of parameters of the detection model according to the current network situation, and realizes the real-time adaptive DDoS detection in a high-speed network. Compared with the previous DDoS attack detection method, the main contributions in this paper are as follows:

- (1) We proposed an adaptive DDoS attack detection algorithm, which can update the model adaptively according to the network situation without manually setting the detection threshold parameters in advance.
- (2) We collected high-speed network traffic from the real-world backbone network boundary. In addition, we sampled the network traffic at different rates to make it closer to the real-world network detection environment.
- (3) We evaluated our detection method in comprehensive aspects, including the resource consumption, the real-time performance which rarely appeared in previous work.

The rest of this paper is arranged as follows: Section 2 describes the related work. Section 3 introduces the attack detection method. Section 4 is the experiment and verification. Section 5 is the summary and prospect.

2. Related Work

DDoS attack detection is different from the deployment of detection points, which can be divided into source detection,

intermediate network detection, and victim detection. Some of the work is summarized as follows:

- (1) For the scene of DDoS attack detection at the attack source, Mergendahl et al. [7] proposed an improved FR-WARD method based on D-WARD for IoT environment, which can accurately detect and defend DDoS attacks and reduce the retransmission overhead of benign IoT devices. Tang et al. [8] proposed a framework FDDA for fast detection and defense of DDoS attacks in the web application environment. They used the DBSCAN method to establish the blacklist in the scanning stage, which makes attack mitigation faster. Biswas et al. [9] proposed a DDoS attack detection method based on behavior similarity between virtual machines for DDoS attacks in the data center.
- (2) For the scene of DDoS attack detection at the victim end, Rahmani et al. [10] proposed a statistical method based on network anomaly and joint entropy of multiservice distribution, which judges the occurrence of attacks by measuring the statistical correlation between the time series of the number of IP flows and the total traffic size. Compared with some methods only using the traffic size, the method has fewer false positives. Mallikarjunan et al. [11] used PCA to reduce the dimension of features and tested the accuracy of machine learning algorithms such as naive Bayes, j48, and random forest on the data set created by the author. The results show that the performance of naive Bayes is better. Aamir et al. [12] used a semisupervised machine learning method to cluster the data using traffic rate, processing delay, and CPU utilization information collected by the victim.
- (3) For the scene of DDoS attack detection at the intermediate network, Barati et al. [13] proposed a DDoS attack detection algorithm based on hybrid machine learning. The method uses a genetic algorithm to select features and the multilayer perception (MLP) in ANN to detect attacks. The accuracy of the algorithm is higher than that of the simple machine learning algorithm. Yusof et al. [14] proposed a method of attack detection of PTA-SVM, by combining SVM with data packet threshold algorithm (PTA). Compared with the improved k-means and logistic regression technology, the PTA-SVM method has a smaller false alarm rate and higher accuracy.

Attack detection in ISP level large-scale network environments is a typical example of intermediate network detection. Compared with the other two attack scenes, more network traffic data can be obtained in intermediate network detection, which makes the detection more accurate and flexible. However, at the same time, the network traffic collected in the intermediate network is larger and the network flow rate is faster, which puts forward higher requirements for the feature storage and calculation.

Many researchers focus on sampling technology for the measurement and statistics of the high-speed network. Ujjan et al. [15] used deep learning, with sFlow sampling and adaptive polling sampling, to detect DDoS attacks. Biswas et al. [9] proposed a flow grouping method based on the behavior similarity between virtual machines and combined with the optimization solver to specify a better sampling rate. The main work of our paper is to focus on the use of light features and based on sketch to achieve high-speed network traffic processing.

In the implementation of DDoS attack detection methods, most of the methods are based on machine learning. Zekri et al. [16] proposed an attack detection algorithm based on decision tree in the cloud environment. Both Hou et al. [17] and Filho et al. [18] used the random forest method to identify attacks. The method proposed by Idhammad et al. [19] combines entropy estimation with Extra-Trees to detect DDoS attacks.

In addition, some researchers have compared different machine learning methods. For example, Priya et al. [20] used three classification algorithms KNN, Random Forest, and Naive Bayesian to detect DDoS Attacks based on the features of incremental time and packet size. Saini et al. [21] used random forest algorithm, Naive Bayes algorithm, and j48 algorithm to detect attacks, and the j48 algorithm produced the best results.

There are also some works based on deep learning. The method proposed by Doshi et al. [22] uses a combination of deep learning and support vector machine to detect attacks. Yuan et al. [23] proposed a DDoS attack detection method based on the recurrent neural network (RNN).

Since most of these methods are supervised or semi-supervised, it is time-consuming to training the classifier on a large amount of network traffic data. Therefore, real-time detection is not guaranteed if the algorithm is deployed in a high-speed network.

Given above, we propose a real-time sketch-based adaptive DDoS detection method. We address more practical issues in real-world detection, such as real-time performance and adaptive ability in the high-speed network environment.

3. Real-Time Sketch-Based Adaptive DDoS Detection

In this paper, we designed an adaptive DDoS attack detection method named RT-SAD, which is based on the asymmetry of network traffic when DDoS attacks occur.

This section is divided into four parts. Firstly, we will describe the overall framework of the detection method. Secondly, we will explain the principle of attack detection. Finally, we will introduce the realization of two core functions: feature statistics and model updating.

3.1. Overview. The overall architecture of this DDoS detection system is shown in Figure 1. The system is mainly composed of the feature statistics module, the attack

detection module, and the model updating module, which is implemented based on sketches.

In the detection process, multiple flow records in fixed time intervals will form a time window. When each flow record in the time window arrives, it will go through the feature statistics module first. Two sketch tables in the module work together to realize the statistics and update asymmetric flow features. After the feature statistics, the attack detection module will use three sketch tables to detect the attack. The three tables used in the detection module are dynamically updated. After the detection module detects all the flow records in a time window, the model updating module will start to work. The module updates the predictive value and threshold of the current window by learning the features of the history window. The predictive value and threshold used in the next time window for attack detection are the updated predictive value and threshold.

The meanings and functions of the five sketch tables in the detection system are shown in Table 1.

In the next part of the article, we will introduce the principle of attack detection in detail.

3.2. Attack Detection. In the network communication model of client-server, there should be both requests and responses. When the server suffers a DDoS attack, the request traffic sent by the botnet will be much larger than the response traffic returned by the server. Because the attacker wants to exhaust the resources of the server as much as possible, the network traffic between clients and the server will show the phenomenon of asymmetry.

In order to quantify the asymmetry of network flow, we propose a quantitative method of asymmetry. We use a pair of IP addresses to represent the flow record. As shown in Figure 2, there are bidirection data transfers between IP-A and IP-C, so it is considered that the request from A to C is normal. As for IP-B and IP-C, there is only traffic from B to C and no traffic from C to B, it is considered that traffic between IP-B and IP-C is asymmetric. And in the current time window, the asymmetric flow feature of IP-C will be increased by 1.

After the analysis of real network traffic, we found that when a DDoS attack occurs, the victim server usually cannot respond to all the clients. There will be a large number of one-way traffic whose destination address is the victim host. That is, when a DDoS attack occurs, the value of asymmetric feature corresponding to some IP addresses will be significantly higher than the normal situation, as shown in Figure 3. In this paper, we mainly use asymmetric of traffic to detect DDoS attacks.

The complete attack detection process is shown in Figure 4. There are two important parts during the detection process. One is the feature statistics and attack detection when each flow record arrives, and the other is the model updating process, including predicted value update and threshold update at the end of the current time window.

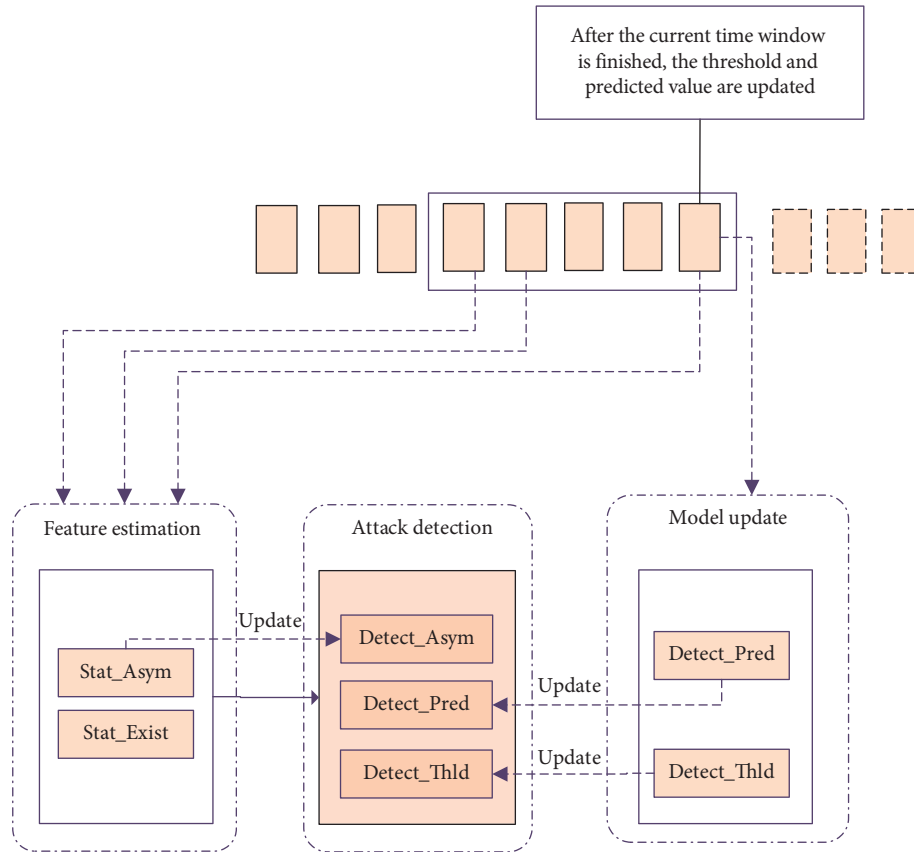


FIGURE 1: Overall architecture.

TABLE 1: Function of sketch.

Sketch name	Meaning	Function
Stat_Asym	The asymmetric flow features of destination IP	To complete the statistical function of feature in a single time window
Stat_Exist	The existence of SIP and DIP	
Detect_Asym	The feature value in the attack detection window	To provide the judgment conditions for the attack detection module to work
Detect_Pred	The predicted value in the attack detection window	
Detect_Thld	The threshold value in the attack detection window	

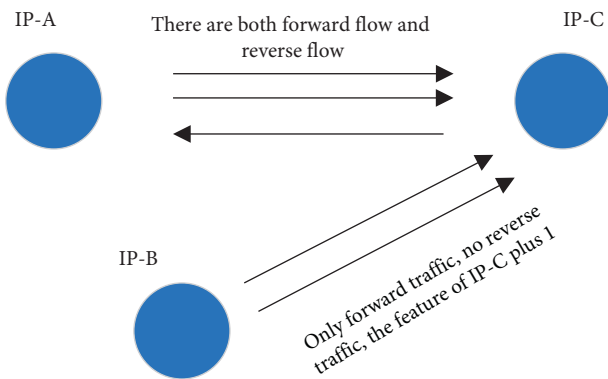


FIGURE 2: Definition of asymmetric flow.

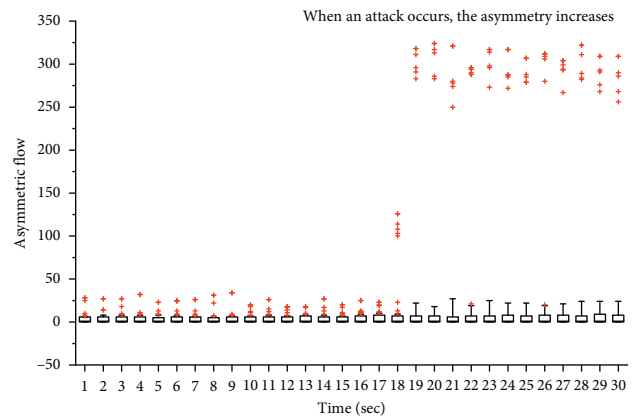


FIGURE 3: Asymmetric feature when the attack occurs.

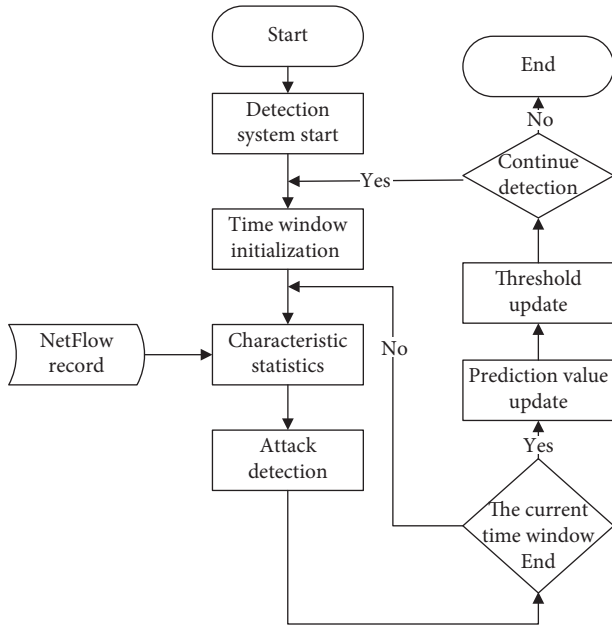


FIGURE 4: DDoS attack detection process.

The two important parts are as follows:

- (1) When a flow record (SIP and DIP) arrives at the detection system, the system will first update the feature corresponding to the DIP in the current window. And then the system will use the feature of DIP, the predicted value, and the threshold calculated according to the feature in the history window, to identify whether the destination IP address in the current flow record is suffering from DDoS attacks. The system will give an attack warning of the victim IP if the detection result is true.
- (2) At the end of the current time window, the system will update the predicted value and threshold of the feature corresponding to the flow records, and the update is only for the normal IP without attack warning, while the feature of attacked IP will not be updated until they return to normal.

The specific detection method is shown as Algorithm 1.

The next part of this paper will describe the algorithm and implementation of feature statistics and model updating.

3.3. Sketch-Based Estimation of Asymmetric Flows. As shown in Figure 1, we use two Sketch tables, Stat_Asym and Stat_Exist to record and update the features of asymmetric flows corresponding to IP in the current time window. More specifically, Stat_Asym is used to record the actual asymmetric flow value of each IP, and stat_Exist is used to record the existence of IP pairs (SIP and DIP).

Figure 5 shows the statistics and update rules of the features. When the flow record arrives, the system will update the Stat_Asym according to the existence of IP pairs recorded in Stat_Exist.

More specifically, for arriving flow record (SIP and DIP), the system finds the values of Stat_Exist[SIP|DIP] and Stat_Exist[DIP|SIP], respectively, which represents the existence of two tuples (SIP and DIP) and (DIP and SIP), and updates the current asymmetric feature according to the different existence conditions of these IP pairs. The value of Stat_Exist represents the existence of IP pairs. If Stat_Exist[SIP|DIP] is 0, it means that the traffic corresponding to the tuple (SIP and DIP) has not appeared in this time window. If the value is greater than 0, it means that the traffic corresponding to the tuple (SIP and DIP) has appeared in this time window.

There are four combinations of Stat_Exist[SIP|DIP] and Stat_Exist[DIP|SIP]. In the attack detection process, we mainly focus on whether the destination IP is attacked; that is, we mainly consider the asymmetric feature of DIP, so only in some cases, the system needs to update the Stat_Exist. The specific update algorithm is shown in Algorithm 2.

After the above steps, the statistics and updates of features are completed. Sketch Stat_Asym[DIP] represents the feature value corresponding to DIP in the current time window.

3.4. Model Updating. At the end of the current time window, the detection system will update the predicted value, threshold, and model parameters. In the system implementation, the sketch table Detect_Thld is responsible for the storage of threshold, and the calculation of threshold is related to the sequence of historical residuals ($res_1, res_2, \dots, res_n$). The residuals of an IP in the time window m , res_m , means the difference between the feature value and the predicted feature value.

The threshold corresponding to an IP in the current window is calculated by three-sigma rule, as shown in the following equation:

$$\text{threshold} = \text{mean}(\text{residual}) + 3 * \text{std_dev}(\text{residual}), \quad (1)$$

where residual refers to the sequence of historical residuals ($res_1, res_2, \dots, res_n$). The mean(residual) is the average value of the historical residual sequence. The std_dev(residual) is the standard deviation of the historical residual sequence.

For the storage and update of dynamic threshold, if the historical values of residuals corresponding to all IP in the past n windows are completely recorded and then the mean and variance of residuals are calculated, too much storage space will be consumed. Therefore, in order to save resources as much as possible, the residual values in all the latest n historical time windows are not directly recorded, but the threshold values are updated by rolling update. And the variance and mean values in multiple historical windows are replaced by progressive variance and mean values. In this paper, the online mean and variance algorithm proposed by Welford [24] is used. The specific formula is shown as

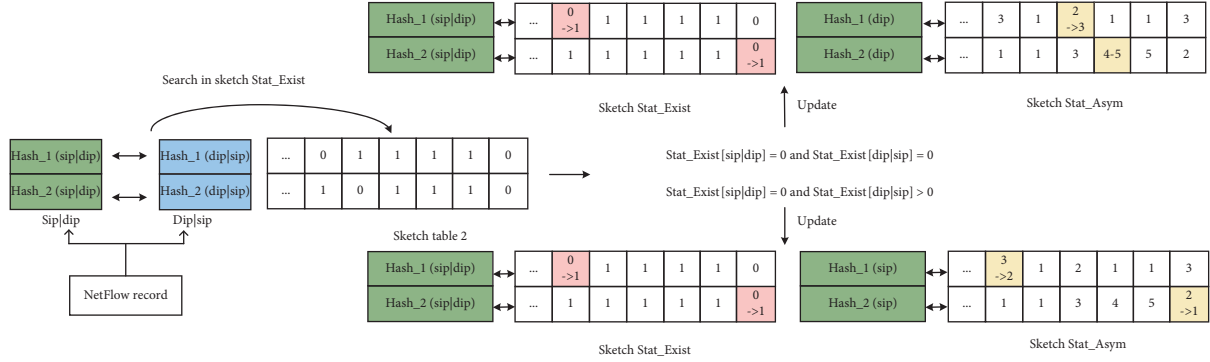


FIGURE 5: Estimation of asymmetric flows.

$$\text{mean}_n(X) = \text{mean}_{n-1}(X) + \frac{x - \text{mean}_{n-1}(X)}{n}, \quad (2)$$

where X is a random variable, x is the n th number of X , and $\text{mean}_n(X)$ represents the mean of the first n numbers in X .

The progressive variance calculation method is shown as

$$\begin{aligned} \text{Var } D_n(X) &= \text{Var } D_{n-1}(X) \\ &+ (x - \text{mean}_{n-1}(X))(x - \text{mean}_n(X)), \end{aligned} \quad (3)$$

$$\text{Var}_n(X) = \frac{\text{Var } D_n(X)}{n}, \quad (4)$$

where X is a random variable, x is the n th number of X , $\text{mean}_n(X)$ represents the mean of the first n numbers in X , and $\text{Var}_n(X)$ represents the variance of the first n numbers in X .

In order to improve the calculation accuracy, only the intermediate value $\text{Var } D_n(X)$ is recorded, which will be used to calculate the variance value. Therefore, we only need to record the progressive mean value, the progressive variance median value, and the number of cycles to update the mean value and variance.

In the process of threshold calculation, the predicted value used in the calculation also needs to be updated adaptively. The system uses the table Detect_Pred to record and update the predicted value. The update rule adopts a simple and efficient single exponential smoothing method, as shown in

$$\begin{aligned} \text{predAsymVal}_{\text{new}}[\text{IP}] &= (1 - \alpha) * \text{predAsymVal}_{\text{old}}[\text{IP}] \\ &+ \alpha * \text{currAsymVal} \end{aligned} \quad (5)$$

where $\text{predAsymVal}_{\text{old}}[\text{IP}]$ is the predicted value of the old asymmetric flow number features of the current IP, and the new one is $\text{predAsymVal}_{\text{New}}[\text{IP}]$; currAsymVal is the number of asymmetric flows corresponding to the IP in the current window.

The value of parameter α in the single exponential smoothing formula is usually set to a specific value between 0.3 and 0.7, but this setting method does not take into account the changes of the current traffic and detection situation. The method in the paper updates α by learning the

historical traffic by adopting a specific strategy. The specific parameter value update algorithm is shown in Algorithm 3.

The above strategy, used to update the α parameter, can make the system recover as soon as possible after the occurrence of false positives, to reduce the possibility of continuous false positives caused by one false positive.

In the current system, we set $\alpha_{\min} = 0.3$, $\alpha_{\max} = 0.7$, and $\Delta\alpha = 0.1$.

4. Experiment and Evaluation

4.1. Dataset. The experiment dataset is a mixture of the background network traffic collected from the real-world backbone network and the attack traffic generated by stress testing tool:

- (1) In the mixed traffic, the background traffic data are collected from the CERNET backbone network for 60 minutes. In addition, we only intercepted the first 64 bytes of each packet. The intercepted data are about 83 GB, the total amount of original data is about 1373 GB, and the actual flow rate is about 3 Gbps.
- (2) The attack traffic data in the mixed traffic are composed of 13 kinds of network layer and transport layer DDoS attack traffic, including UDP Flood, UDP Fragmentation Flood, ICMP Flood, ICMP Fragmentation Flood, TCP SYN Flood, TCP-SYN ACK Flood, TCP ACK Flood, TCP ACK Fragmentation Flood, TCP PUSH ACK Flood, TCP RST Flood, TCP FIN Flood, TCP URG Flood, and X_MAS Flood. The attack traffic is generated by stress testing tool and mixed with normal background traffic at different time points. The scale of each attack traffic is about 1 Gbps, and the IP number of the attack target is 5.

The network flow speed under different sampling rates in this experiment is shown in Table 2.

4.2. Evaluation Criteria. Our solution uses the sliding window method in the detection process, so we use the time window as the unit to evaluate our experiment result.

```

Input: NetFlow Record, Sketch Detect_Asym, Detect_Pred, Detect_Thld
Output: DDoS attack detection results
(1) while Current window not end do
(2) Get a NetFlow record (SIP, DIP)
(3) Update the Detect_Asym [DIP] by using (SIP, DIP)
(4) Residual = abs (Detect_Pred [DIP] – Detect_Asym [DIP])
(5) if Residual > Detect_Thld[DIP] then
(6)     Alert of DDoS
(7) else
(8)     Put the DIP into Update_Set
(9) end if
(10) end while
(11) for each DIP in Update_Set do
(12) Update Detect_Pred [DIP]
(13) Update Detect_Thld[DIP]
(14) end for

```

ALGORITHM 1: DDoS attack detection algorithm.

In the whole attack detection process, there are four kinds of detection results corresponding to the actual data for the current window, as shown in Table 3.

In each subsequent experiment, we use three indicators to evaluate the effectiveness of the method, namely, accuracy rate (AR), false positive rate (FPR), and false negative rate (MR).

4.3. Resource Evaluation. In this paper, the sampling technique and probability data structure in high-speed network measurement are used to optimize the cost of storage and computing resources. In order to evaluate the resource cost of the proposed algorithm, we conducted two experiments, sampling rate experiment and sketch size experiment. The sketch experiment is to evaluate the detection performance of the algorithm when using different sizes of sketch data structures. The sampling rate experiment evaluates the detection performance of the algorithm for network traffic data with different sampling rates.

The size of the sketch will have an impact on the accuracy of the detection algorithm. Therefore, our sketch resource consumption experiment mainly compares the detection performance of the algorithm by setting a fixed sampling rate and selecting different sizes of the sketch. The sampling rate of the experiment data is 10 : 1, and the size of the sliding window is 1 second. Under this configuration, the number of flows per second is about 9000 without attack and 17,000 under attack. Therefore, the sketch size is set to the following five groups, ranging from 2^{12} (4K) to 2^{15} (32K). The experiment results are shown in Table 4.

Under this condition, when the sketch size is 2^{13} or higher, the algorithm can achieve better results. At the same time, if we want to get better performance, we need to consume more storage resources. In practice, we need to select an appropriate sketch size according to current network flow speed and current hardware performance.

In the case of a high-speed network, the performance of the detection algorithm not only depends on the complexity of its own algorithm but also has a great relationship with the

current flow speed. In order to make the network flow speed match the processing flow speed of the algorithm as much as possible, we conducted four groups of experiments using the mixed network flow with different sampling rates: 10 : 1, 20 : 1, 100 : 1, and 200 : 1.

The size of the sketch is 2^{15} , and the size of the detection window is 1 second. Table 5 shows the performance results of the detection algorithm under different sampling rates.

It can be seen from the experiment results that when the algorithm configuration is appropriate, it has good detection performance for different sampling rates of network traffic.

4.4. Real-Time DDoS Detection. In order to evaluate the real-time performance of the algorithm for DDoS attack detection, we design a real-time evaluation experiment. We take the time from attack occurrence to detection algorithm alarm as the experiment measurement criteria. We tested the detection time of the current algorithm for different sampling rates, and the experiment results are shown in Table 6.

From the experiment results, it can be seen that in the current experiment, the algorithm has good real-time performance for the different sampling rates of network traffic. In addition, as the sampling rate increases, the number of flows per unit time decreases, so the processing efficiency of the algorithm increases and the detection time decreases.

4.5. Results on Different DDoS Attack Detection. In order to evaluate the applicability of our algorithm for different DDoS attacks, we designed an attack detection applicability experiment. In this experiment, we generate attack traffic for each attack and then mix it into the background traffic separately to detect the performance. The results of the performance for different attacks are shown in Table 7, with the sketch size of 2^{15} and sampling rate of 10 : 1.

From the experiment results, it can be seen that the algorithm has higher detection accuracy for different types of the network layer and transport layer DDoS attacks with lower false alarm rate and missing alarm rate.


```

Input: Sketch Stat_Exist, Stat_Asym
Output: Sketch Stat_Exist, Stat_Asym
(1) if Stat_Exist[SIP|DIP] = 0 then
(2)   Stat_Exist[SIP|DIP] = Stat_Exist[SIP|DIP] + 1
(3)   if Stat_Exist[DIP|SIP] = 0 then
(4)     Stat_Asym[SIP] = Stat_Asym[SIP] + 1
(5)   end if
(6)   if Stat_Exist[DIP|SIP] > 0 then
(7)     Stat_Asym[SIP] = Stat_Asym[SIP] - 1
(8)   end if
(9) end if

```

ALGORITHM 2: Feature updating algorithm.

```

Input: Current residual res, Historical residual sequence res_list,  $\Delta\alpha$ ,  $\alpha$ ,  $\alpha_{\min}$ ,  $\alpha_{\max}$ 
Output:  $\alpha$ 
(1) if res > mean(res_list) + 3 * std_dev(res_list) then
(2)   if  $\alpha < \alpha_{\max}$  then
(3)      $\alpha = \alpha + \Delta\alpha$ 
(4)   end if
(5) else if res < mean(res_list) - std_dev(res_list) then
(6)   if  $\alpha > \alpha_{\min}$  then
(7)      $\alpha = \alpha - \Delta\alpha$ 
(8)   end if
(9) end if

```

ALGORITHM 3: Parameter value α update algorithm.

TABLE 2: Flow speed with the different sampling rates.

Sampling rate	Flow speed/flows per second	
	Without attack	Under attack
10:1	9476	17,418
20:1	5403	9382
100:1	1351	2151
200:1	727	1132

TABLE 3: Confusion matrix.

		Algorithm detection results in the current window	
		Attack detected	No attack detected
Actual situation in current window	Under attack	Correct (TP)	Failing to report (FN)
	Without attack	Wrong report (FP)	Correct (TN)

TABLE 4: Performance of different sketch sizes.

Sketch size	AR (%)	FPR (%)	MR
2^{12}	89.87	14.03	0
2^{13}	96.89	4.34	0
2^{14}	99.82	0.24	0
2^{15}	99.77	0.32	0

TABLE 5: Performance of different sampling rates.

Sampling rate	AR (%)	FPR (%)	MR (%)
10:1	99.77	0.32	0
20:1	99.77	0.12	0.48
100:1	99.11	0	2.98
200:1	99.88	0	0.39

TABLE 6: Real-time performance of the algorithm.

Sampling rate	Detection time/millisecond
10:1	1064
20:1	1029
100:1	232
200:1	122

TABLE 7: Performance of different DDoS attacks.

The type of DDoS attack	AR (%)	FPR (%)	MR (%)
SYN Flood	99.48	0.59	0.29
SYN-ACK Flood	99.60	0.32	0.58
ACK Flood	99.40	0.28	1.37
ACK&PUSH Flood	99.74	0.36	0
RST Flood	99.57	0.35	0.59
FIN Flood	99.68	0.20	0.56
TCP URG Flood	99.71	0.80	0.77
X_MAS Flood	99.74	0.20	0.37
ACK Fragmentation Flood	99.57	0.28	0.77
UDP Fragmentation Flood	99.48	0.32	0.96
ICMP Fragmentation Flood	99.63	0.08	1.04
ICMP Flood	99.60	0.44	0.29
UDP Flood	99.45	0.28	1.15

5. Conclusions

Given the current threat of DDoS attacks, we propose a real-time DDoS attack detection method based on sketch for intermediate networks. In this paper, the sketch is used to record and update the features which are needed for attack detection, and the adaptive threshold of the feature is dynamically updated by the historical network traffic. The experiment results show that the method has good performance in accuracy, resource consumption, and real-time performance. At the same time, there are still some improvements in this method, such as adaptive network traffic sampling and adaptive size adjustment of sketch structure changed with the network situation. This is also the content of our following work.

Data Availability

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

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the joint fund of the Ministry of Education of China and China Mobile (MCM20180506).

References

- [1] K. Abbas, L. A. Tawalbeh, A. Rafiq, A. Muthanna, I. A. Elgendy, and A. A. Abd EI-Latif, "Convergence of blockchain and IoT for secure transportation systems in smart cities," *Security and Communication Networks*, vol. 2021, Article ID 5597679, 13 pages, 2021.
- [2] T. Emmons, "Part I: retrospective 2020: DDoS was back-bigger and badder than ever before," 2021, <https://blogs.akamai.com/2021/01/part-i-retrospective-2020-ddos-was-back-bigger-and-badder-than-ever-before.html>.
- [3] L. Jakober, "Akamai mitigates sophisticated 1.44 Tbps and 385 Mpps DDoS attack," 2020, <https://blogs.akamai.com/2020/06/akamai-mitigates-sophisticated-144-tbps-and-385-mpps-ddos-attack.html>.
- [4] C. Cimpanu, "FBI warns of new DDoS attack vectors: CoAP, WS-DD, ARMS, and Jenkins," 2020, <https://www.zdnet.com/article/fbi-warns-of-new-ddos-attack-vectors-coap-ws-dd-arms-and-jenkins/>.
- [5] Kalitool, "Hping3 package description," 2019, <https://tools.kali.org/%20information-gathering/hping>.
- [6] abatishchev, "Loic a network stress testing application," 2019, <https://sourceforge.net/projects/loic/>.
- [7] S. Mergendahl, D. Sisodia, J. Li, and H. Cam, "Source-end DDoS defense in IoT environments," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, Dallas, TX, USA, November 2017.
- [8] T. M. Thang and K.-V. Nguyen, "FDDA: a framework for fast detecting source attack in web application DDoS attack," in *Proceedings of the Eighth International Symposium on Information and Communication Technology*, Nha Trang, Vietnam, December 2017.
- [9] R. Biswas, S. Kim, and J. Wu, "Sampling rate distribution for flow monitoring and DDoS detection in datacenter," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2524–2534, 2021.
- [10] H. Rahmani, N. Sahli, and F. Kammoun, "Joint entropy analysis model for DDoS attack detection," in *Proceedings of the Fifth International Conference on Information Assurance and Security*, Xi'an, China, August 2009.
- [11] K. Narasimha Mallikarjunan, A. Bhuvaneshwaran, K. Sundarakantham, and S. Mercy Shalinie, "DDAM: detecting DDoS attacks using machine learning approach," *Computational Intelligence: Theories, Applications and Future Directions-Volume I*, Springer, New York, NY, USA, 2017.
- [12] M. Aamir and S. M. Ali Zaidi, "Clustering based semi-supervised machine learning for DDoS attack classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 2, 2019.
- [13] M. Barati, A. Abdullah, N. I. Udzir, R. Mahmud, and N. Mustapha, "Distributed Denial of Service detection using hybrid machine learning technique," in *Proceedings of the International Symposium on Biometrics and Security Technologies (ISBAST)*, Kuala Lumpur, Malaysia, August 2014.
- [14] M. A. M. Yusof, H. M. A. Fakariah, and Y. D. Mohamad, "Detection and defense algorithms of different types of DDoS attacks," *International Journal of Engineering and Technology*, vol. 9, p. 410, 2017.
- [15] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.
- [16] M. Zekri, S. E. Kafhali, N. Aboutabit, and Y. Saaadi, "DDoS attack detection using machine learning techniques in cloud computing environments," in *Proceedings of the 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, Rabat Morocco, October 2017.
- [17] J. Hou, P. Fu, Z. Cao, and A. Xu, "Machine learning based DDoS detection through NetFlow analysis," in *Proceedings of*

- the MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, Los Angeles, CA, USA, October 2018.
- [18] F. S. d. L. Filho, F. A. F. Silveria, A. d. M. B. Junior, G. Vargas-solar, and L. F. Silverira, "Smart detection: an online approach for DoS/DDoS attack detection using machine learning," *Security and Communication Networks*, vol. 2019, Article ID 1574749, 15 pages, 2019.
 - [19] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for DDoS detection," *Applied Intelligence*, vol. 48, no. 10, pp. 3193–3208, 2018.
 - [20] S. S. Priya, M. Sivaram, D. Yuvaraj, and A. Jayanthildevi, "Machine learning based DDoS detection," in *Proceedings of the International Conference on Emerging Smart Computing and Informatics (ESCI)*, Pune, India, March 2020.
 - [21] P. S. Saini, S. Behal, and S. Bhatia, "Detection of DDoS attacks using machine learning algorithms," in *Proceedings of the 7th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, March 2020.
 - [22] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer internet of things devices," in *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, May 2018.
 - [23] X. Yuan, C. Li, and X. Li, "DeepDefense: identifying DDoS attack via deep learning," in *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, Hongkong China, May 2017.
 - [24] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419-420, 1962.