Hindawi
Security and Communication Networks
Volume 2022, Article ID 3161143, 14 pages
https://doi.org/10.1155/2022/3161143

Research Article

A Protocol-Independent Botnet Detection Method Using Flow Similarity

Jianbing Liang, Shuang Zhao, and Shuhui Chen

College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Shuhui Chen; shchen@nudt.edu.cn

Received 31 December 2021; Accepted 23 June 2022; Published 30 July 2022

Academic Editor: Marimuthu Karuppiah

The detection of botnets has always been a hot spot in the field of network security. However, there are still many challenges in botnet detection. Most of the current botnet detection approaches, such as machine learning and blacklists, cannot discover evolving botnet variants. These methods are usually only valid for specific botnet protocols which are not general. Even they may be difficult to deal with encrypted botnet traffic. In this paper, we design a protocol-independent botnet detection method for these challenges. Our detection method takes advantage of the group characteristic of the botnet, which is the inherent characteristics of the botnet. We use the sequence of packet length as the characteristic of a flow. Then, we calculate the similarity between these sequences to detect botnets. Our method has an excellent generality, which is not affected by encrypted traffic and the protocols of the botnet. Experiments on a challenging dataset ISCX show that the proposed method can effectively detect botnets with a high average detection rate and low false alarm, which significantly outperforms the state-of-the-art methods. Therefore, the proposed detection method is robust and has a wide range of adaptability in detecting botnets.

1. Introduction

A botnet is a one-to-many network formed between the controller and the infected host. There are many methods that can be used by botnet controllers (attackers) to spread bot viruses. Once the host is infected with a bot virus, it will become a part of this botnet. The infected host will receive the attacker's instructions through a control and command (C&C) channel. The infected computers (bots) are silently driven and commanded by the botnet controller to launch cyberattacks. A botnet is equivalent to a platform for attackers to control bots to perform malicious activities. Attackers can conduct distributed denial of service (DDoS) attacks, spread spam, perform network blackmail, and steal personal information through botnets. It brings great challenges to network security and personal privacy protection. Hosts infected as bots can avoid being discovered by network monitoring agencies in a variety of methods, such as constantly updating themselves, disabling antivirus applications, and preventing DNS from looking up certain domain names. These methods increase the difficulty of botnet detection.

It is well known that the threat of botnets to the Internet is exceedingly scary. With the development of new technologies, botnet detection is facing increasing challenges. Mirai is a new type of botnet that has emerged in recent years. It is the driving force behind the latest large-scale DDoS attack [1]. Mirai infects more than 100,000 IoT devices to form a huge botnet, which may be the largest DDoS attack in history. It is estimated that Mirai's throughput has reached 1.2 Tbps.

The structure of botnet can be summarized into two categories, namely, centralized and decentralized structure. For a centralized botnet, a communication channel is established between the C&C server and all bots. There are many botnets that are based on centralized structure, such as AgoBot, SDBot, and RBot [2, 3]. The protocols adopted by these botnets are mainly based on HTTP and Internet Relay Chat (IRC) protocols. The flexible and simple structure of the IRC protocol is favored by many hackers. Botnets based on the HTTP protocol are usually concealed and difficult to

detect. The decentralized botnet uses P2P-based protocols. When issuing the command, the botmaster randomly selects a bot as the C&C server to communicate with other bots. Since the P2P-based botnets effectively avoid the problem of a single point of failure, they greatly enhance the survivability of the botnet [4].

The botnet detection technology has always been a research hotspot in the field of network security. Researchers have proposed a large number of methods to detect botnet [5, 6]. These methods can be summarized into five categories, that is, signature-based methods, anomaly-based methods, honeypot-based methods, specific protocol structure-based methods, and community-based methods. The signature-based methods [7] cannot detect unknown botnets and their variants. Moreover, the encryption technology used in the botnet negates the effects of these methods. Anomaly-based detection methods [8] are based on the assumption that the communication pattern of the botnet is different from that of the benign network. However, the bots can mimic the communication pattern of the normal hosts to evade the anomaly detection technology. Detection methods based on honeypot technology can only detect existing botnets. This method has poor real-time performance. Detection methods based on specific protocols and structures [9] cannot detect botnets with different protocols or structures. For community-based anomaly detection algorithms [10], they cannot accurately identify botnets when there is no complete communication graph. Nowadays, cyberhackers are adopting new technologies to constantly update botnets in terms of creation, maintenance, and communication mechanisms. Therefore, existing detection technologies cannot cope with unknown and increasingly complex botnets.

A botnet is defined as a coordinated group of malware instances that are controlled by a botnet master via C&C channels [11]. The bots in the same botnet have the same or similar traffic characteristics. In this paper, we propose a protocol-independent botnet detection framework to identify botnet traffic by analyzing the similarity of the traffic flows. Our method can discover bots who initiate these flows that have similar traffic characteristics. More specifically, if the network traffic initiated by a certain host has a great similarity, it can be concluded that the traffic is generated by botnet activities according to the attributes of the botnet. The hosts involved in this traffic are bots in the monitored network. We use the sequence of packet length as the characteristic of the flow. The sequence of packet length is easy to obtain and is very effective for detecting botnets. The sequence of packet length is a vector composed of the length of all the packets in a flow. Each element in the sequence is arranged in sequence according to the order of packet transmission. The degree of similarity between these flows determines whether these flows are botnet traffic. In addition, although the length of the ciphertext output by the encryption algorithm may be different from that of the plaintext, the length of the ciphertext output by the same encryption algorithm is the same for the plaintext of the same length. Therefore, for the packets in a network flow, the encryption algorithm will not change the relationship between the lengths of these packets. Hence, the length of packets applied as the characteristic of the flow makes the detection method very robust.

This paper makes the following major contributions:

(i) A protocol-independent botnet detection framework is proposed based on the group characteristics of botnets, which are the inherent characteristics of botnets. Our botnet detection framework is not affected by the C&C protocol. It can be applied to detect bots in both centralized and P2P-based botnets. Compared with the prior work, the detector proposed in this paper is always reliable and efficient, no matter what C&C protocol the botnet adopts.

(ii) The sequence of packet length is proposed as the characteristic of the flow, which is easy to obtain and is effective for detecting botnets. The packet length applied as the characteristic of the flow makes the detection method very robust.

(iii) A bot detection prototype system is implemented. The detection effect of the system is evaluated on dataset ISCX [12]. The results show that the system has a high true positive rate and a low false positive rate.

2. Related Work

Many researchers have been making continuous efforts to detect botnet. BotMiner [11] is a framework to detect groups of compromised machines that are part of a botnet. The framework is independent of the C&C protocol. It identifies bots by clustering similar malicious traffic and communication patterns. The authors implement the BotMiner prototype system and evaluate the result using traces of many real-world networks. The results show that BotMiner can detect real-world botnets (such as P2P-based botnets, HTTP-based botnets, and IRC-based botnets) with high accuracy and low false positive rate [11]. However, BotMiner needs to analyze the content of the traffic load, which may fail when the traffic is encrypted.

An adaptive framework for detecting botnets is presented in [13]. It is composed of three components, namely, Behavior Extractor, Behavior Identifier, and Feedback Provider. Behavior Extractors generate Behavior Instances (BIs) of hosts from network traffic periodically. BIs are representations of host behavior in a time period. To classify malicious BIs, Behavior Identifier is implemented, which employs a real-time statistical model named Behavioral Model (BM). Feedback Provider can alert the network administrator when it receives a message that malicious BIs are found by Behavior Identifier. At the same time, the Feedback Provider can update BM based on whether the administrator confirms that the host found by Behavior Identifier is malicious. When a new bot appears, the framework requires the administrator to confirm whether the bot is genuine. Therefore, the professional level of the administrator may be the bottleneck that affects its detection of new bots.

DBod is a DGA-based botnet detection framework based on analysis of the query behavior of DNS traffic [14]. The research assumes that bots in the same DGA-based botnet

query the same sets of domains in the domain list. Since only a very limited number of the domains are actually associated with an active C&C communication [14], most DNS requests sent by bots will fail and generate NXDomains. The main observation behind DBod is that DGA-based bots are different from benign hosts in the distribution of DNS query time and the count of NXDomains. DBod consists of a filtering module, a clustering module, and a group identification module [14]. DBod does not require prior knowledge for training and can detect new bots. However, it will fail when there is no DNS traffic.

Wang et al. [15] propose a two-stage approach for botnet detection. In the first stage, they perform two different anomaly detection, namely, flow-based anomaly detection and graph-based anomaly detection. In the second stage, they identify the pivotal nodes of the discovered anomalies, evaluate pivotal interaction measures, and construct correlation graphs. Community detection is used to identify botnets. Their approach is based on two observations: (1) Botmasters and victims communicate with many other nodes, which are easy to be detected. (2) The infected hosts often communicate with each other, resulting in a strong correlation between them.

PsyBoG [16] applies signal processing technology to botnet detection. They analyze the time phase and similarity of DNS traffic to identify botnet. PsyBoG uses power spectral density (PSD) analysis, which is a signal processing technology, to detect the major frequency of periodic botnet behavior. Then, it clusters the hosts based on the similarity of traffic patterns [16]. PsyBoG detects previously unknown botnets based on the suspicious DNS manner.

Zhuang et al. [17] propose an effective system, Enhanced PeerHunter, to detect P2P-based botnet. Enhanced PeerHunter is based on network flow level community behavior analysis. It is capable of detecting P2P botnets when (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) the botnet traffic is overlapped with legitimate P2P traffic on the same host; and (d) no statistical traffic pattern is known in advance (unsupervised). To detect P2P botnets, Enhanced PeerHunter first detects P2P network traffic. Then, it builds a network flow level mutual contacts graph. Finally, it uses community detection to discover P2P-based botnets.

With in-depth research on machine learning, it is increasingly applied to the detection of botnets.

Carl et al. [18] compare the performance of network classifiers based on different machine learning techniques (such as J48, naive Bayes, and Bayesian) to find the classifier with the highest recognition rate. The result is that a naive Bayes classifier performs best. In addition, the classification sensitivity to the training set size is determined experimentally by them in this paper. Accurate labels are critical, however. Once the labels of the training data are inaccurate, the performance of the classifier will suffer greatly.

Mohammad et al. [19] propose an approach that exploits the reinforcement learning technique to detect infected hosts in a peer-to-peer (P2P) botnet. Specifically, they develop a traffic reduction method to deal with a high volume of network traffic. However, botnets dynamically change their operations through updating after several life cycle stages. Hence, the proposed approach will fail if it is not improved dynamically throughout time.

In addition, Pektas et al. [20] design a framework that combines convolutional and recurrent neural network to identify botnets. The proposed system extracts network flow features, such as duration, size of packets, and other related flow-based features. However, this method usually has weak generalization ability and cannot effectively identify unknown types of botnet traffic.

Mousavi et al. [21] focus on scalability in high-rate network bandwidth. They propose a fully scalable big data framework based on Hadoop to deploy many different kinds of botnet detection methods, including statistics-based methods, machine learning-based methods, and graph-based methods. The experimental results show that the framework can perform well. In addition, the running time of the proposed framework is logarithmic proportional to the volume of the input. Despite its advantages, the framework proposed in this paper has its drawback. It is not affordable for smaller enterprises to provide enough computational resources which are required to install the proposed framework.

Soodeh et al. [22] propose a method based on convolution neural networks and negative selection algorithms to detect botnet. They focus on the activity of incoming packets and detect botnet traffic from them. Alharbi and Alsubhi [23] exploit a graph-based machine learning model to detect botnet traffic. They consider the significance of graph features and develop a generalized model for detecting botnets based on features that are selected using five filter-based feature evaluation measures derived from consistency, correlation, and information theory. Biswas and Roy [24] explore a method to detect botnet traffic using deep learning approaches like Artificial Neural Networks (ANN), Gatted Recurrent Units (GRU), and Long or Short Term Memory (LSTM) model. The proposed method has shown how it can perform against both normal attack data and botnet-specific attack data. Javier et al. [25] focus on the method to increase the performance of botnet traffic classification. They use Information Gain and Gini Importance to select features and evaluate the selected features through performing three models, that is, Decision Tree, Random Forest, and k-Nearest Neighbor. Wan et al. [26] design a multilayer framework to detect botnet traffic. The detection model consists of a filtering module and classification module which exploits machine learning algorithms. Their detection model is based on behavior-based analysis. This research examines the features useful for creating a behavior-based analysis method for detecting botnets in network traffic. The computational complexity of the machine learning-based method is relatively large, which is difficult to deploy in the realistic setting. Moreover, the generalization ability of models based on machine learning is limited and cannot cope with the endless botnets.

In conclusion, these existing studies have some limitations. Some methods can only be effective for botnets that use specific protocols. They cannot detect newly emerging botnets. Moreover, some methods are based on historical

data. Once the botnet variations appear, these methods will be powerless. The detection method proposed in this paper is based on the group characteristics of the botnet, which are inherent characteristics of the botnet. Our method is independent of the botnet protocol and is not affected by encrypted data.

3. Detection Framework and Implementation

3.1. Research Objectives. Our research goal is to find the bot in the monitoring network by analyzing the traffic crossing the boundary of the monitoring network. We exploit the fact that all botnets have group characteristics, and the relationship between the length of packets in a flow will not be affected by the encryption algorithm. Specifically, within a certain period of time, the flows generated by bots in the same botnet are similar. We analyze the similarity of network traffic to detect bots of the botnet. It is commonly known that the communication of most botnets is based on Transmission Control Protocol (TCP) [2], such as Waledac botnet [3], storm botnet [4], Conficker botnet [10], and Zeus botnet [9]. Therefore, the research of our method mainly focuses on TCP flows. The process of normal hosts evolving into bots can be divided into three stages. In the first stage, hosts are infected by botnet malware. In the second stage, hosts receive the command from the botmaster and join the botnet. Finally, hosts initiate a network attack at an appropriate time. The host will show malicious abnormal behavior in the second and third stages. Therefore, the detection method proposed in this paper works during the second and third stages to realize the detection of bots. It cannot be able to recognize the hosts that have just been infected by the malware. In this paper, we do not pay attention to how the host is infected or how the botnet malware is spread. Our research goal is to detect the bots that generate malicious TCP flows in the monitored network.

Our research objectives are as follows:

(i) The bot detection framework is independent of the protocol and structure adopted by botnet channels. Its detection performance is not affected by the botnet protocol and structure.

(ii) The bot detection framework does not need to analyze the content of the traffic payload. Hence, it is not affected by encrypted traffic and will not violate the privacy of network users.

(iii) The bot detection framework can effectively detect botnet traffic and identify bots with a high detection rate and a low false positive rate.

(iv) The bot detection framework must have low complexity. It cannot consume too much computing resources and time.

3.2. Bot Detection Framework. As shown in Figure 1, the bot detection framework includes five modules, that is, network traffic acquirer, preprocessing module, attack flow recognizer, infection flow recognizer, and result integration module.

Formally, we define $F_i = (sip_i, dip_i, \text{sport}_i, dport_i, pktlenseq_i)$ to denote the TCP flow with the sequence of packet length of host h_i, where sip_i is the source IP address, sport_i is the source port number, dip_i is the destination IP address, and $dport_i$ is the destination port number. $pktlenseq_i$ is the sequence of packet length, which is a vector composed of the length of all the packets in a flow, as defined in (1). Each element in the sequence is arranged in sequence according to the order of packet transmission. The degree of similarity between flows determines whether the monitored traffic is botnet traffic. Although the length of the ciphertext output by the encryption algorithm may be different from that of the plaintext, the length of the ciphertext output by the same encryption algorithm is the same for the plaintext of the same length. Therefore, for the packets in a network flow, the encryption algorithm will not change the relationship between the lengths of these packets. Therefore, our method based on the packet length for detecting botnets is robust.

$$pktlenseq_i = [\text{len}_{i1}, \text{len}_{i2}, \ldots, \text{len}_{in}]. \tag{1}$$

Suppose there are two flows F_i and F_j, $R(F_i, F_j)$ refers to the communication relationship between F_i and F_j, as defined in (2). The communication relationship indicates whether the two flows have the same mapping of source IP address or destination IP address. If there is a communication relationship, $R(F_i, F_j) = \text{True}$. Otherwise, $R(F_i, F_j) = \text{False}$.

$$R(F_i, F_j) = \begin{cases} \text{True} & \text{if } f(sip_i) = f(sip_j) \\ & \text{or } f(dip_i) = f(dip_j). \\ \text{False} & \text{others.} \end{cases} \tag{2}$$

In (2), $f(ip)$ is the mapping function of IP address. The simplest mapping is self-mapping; that is, $f(ip) = ip$. There are also some other mappings, such as $f(ip) =$ domain name, that is, the mapping relationship between IP address and DNS domain names. If $f(ip_i) = f(ip_j)$, it means that ip_i and ip_j are the same in the "mapping sense." Specifically, if $f(ip) = ip$, then according to $f(ip_i) = f(ip_j)$, it is obvious that $ip_i = ip_j$. If $f(ip) =$ domain name, we can know that ip_i and ip_j have the same domain name and ip_i and ip_j belong to the same host. In this paper, we use the self-mapping, namely, $f(ip) = ip$. FC denotes the set of flows that have communication relationships between each other, as defined in

$$FC = \{F_1, F_2, \ldots, F_k \mid R(F_i, F_j) = 1, 1 \le i, j \le k\}. \tag{3}$$

The network traffic acquirer can be deployed not only inside the monitored network to analyze the traffic in the internal network to detect botnet but also at the boundary of the monitored network. When the network traffic acquirer is deployed at the boundary of the monitored network, it is responsible for capturing the traffic entering and leaving the boundary of the monitoring network. In this case, the traffic captured by the network traffic acquirer is between the internal network and the external network, which does not include pure internal network traffic. The packet lengths are obtained by parsing the IP header of the packet. Then, they

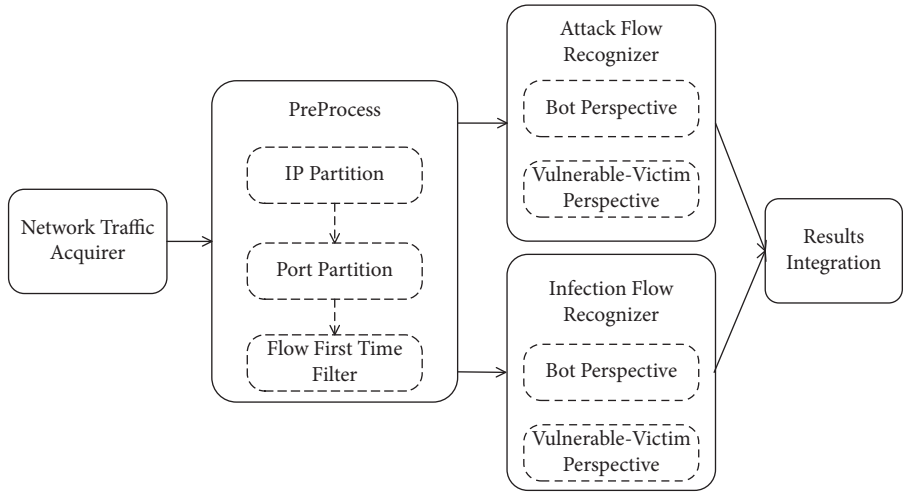

Figure 1: The framework of bot detection.

are integrated into the sequence of packet length in ascending order of the TCP sequence number for flow similarity analysis.

The preprocessing module is composed of three modules, namely, IP Partition, Port Partition, and Flow First Time Filter. Since the bot detection framework is based on the similarity of flows, we are only interested in flows that have communication relationships with each other. Therefore, we must first know which hosts are involved in these flows. The IP Partition module divides the traffic according to whether the collected traffic has a communication relationship (as (3)). It mainly solves the problem of "which hosts communicate with each other." Moreover, the services used for communication between these hosts are also very important. We can determine the services through the TCP port number. The Port Partition module aggregates flows on the same source port number or the same destination port number. It mainly solves the problem of "what communications do the hosts carry out." According to the distribution of port numbers, we divide the flows into two categories, namely, attack flows and infection flows. The attack flows refer to the traffic generated by bots when they launch a network attack. The infection flows refer to the traffic generated by bots when they are in the propagation phase.

We analyze the attack flows and the infection flows from two perspectives, that is, the bot and the vulnerable victim. When a botnet launches a network attack, the vulnerable victims are the attack targets, which may be the target of multiple attacks at the same time. When receiving the attack instruction, the bot will use the maximum resources to launch an attack on the target, such as the traffic of DDoS attacks. Therefore, when observing the attack flows from the perspective of the bot, the distribution of the port numbers presents a many-to-one situation, that is, multiple ports of the bot actively establish TCP connections with the same ports of the vulnerable victims. When observing the attack flows from the perspective of the vulnerable victim, the distribution of port numbers presents a one-to-many situation. The ports of vulnerable victims are passively connected with multiple different hosts.

The infection flows are traffic generated by bots during the process of conducting malware propagation or vulnerability scanning. Meanwhile, some traffic is the commands conveyed by the botmaster to bots. Hence, when observing the infection flows from the perspective of the bot, the unique TCP port of bots actively establishes connections with multiple hosts. From the perspective of the vulnerable victim, the infection flows present that the unique TCP port is passively communicating with the same port of multiple hosts. For each TCP flow, the first packet time of the flow in both directions (upstream and downstream) determines the initiative and passivity of "establishing a TCP flow."

Based on all the above observations, the TCP flows within a certain period of time are grouped according to the IP address and port number to form the TCP flow blocks. Then, the sequences of packet length of the flows in the blocks are obtained. Afterward, the attack flow recognizer calculates the similarity of the packet length sequences of these flows from the perspective of the bot. Meanwhile, the infection flow recognizer calculates the similarity of the packet length sequences of these flows from the perspective of the vulnerable victim. Finally, the result integration module is responsible for summarizing the recognition results of the recognizer and obtains a collection of malicious TCP flows.

The following sections will detail the implementation of each part of the detection framework.

3.3. Network Traffic Acquirer. We have developed an effective network traffic capture module, namely, network traffic acquirer. In this paper, we limit our interest to TCP flows. Each flow contains the following information: source IP, destination IP, source port, destination port, timestamp, and length of packets in two directions. Our research is based on the fact that the TCP flows generated by the bots in the same botnet within a certain time frame are similar. Therefore, we set a flow window according to the start time of the flows. The network traffic acquirer captures a certain number of TCP flows based on the flow window. Let WinSize denote the size of a flow window. When the number of captured flows exceeds WinSize, these

```
Require: packet_len_seq
Ensure: trunc_packetlen_seq
Param: len_limit_1, len_limit_2
  if len(packet_len_seq) > len_limit_1 then
  trunc_packetlen_seq = packet_len_seq[0: len_limit_1]
  end if
  packet_index = 0
  for packet_len ∈ trunc_packetlen_seq do
    if packet_len > 0 then
      packet_index+ = 1
    end if
    if packet_index > = len_limit_2 then
      break
    end if
  end for
  return trunc_packetlen_seq[0: packet_index]
```

Algorithm 1: Truncation algorithm.

flows are submitted as a collection to subsequent modules for analysis to detect malicious flows. WinSize is the minimum number of flows to detect botnet through traffic analysis. In addition, the flow truncation is performed to reduce the computational cost. We empirically use the first 16 packets of the TCP flow rather than the whole TCP flow. If the packet number of the TCP flows is greater than 16, the TCP flow is truncated. The truncation algorithm is shown in Algorithm 1.

The packet_len_*seq*, the input parameter of Algorithm 1, is the sequence of packet length, which is composed of the length of TCP payload. There are two thresholds that have been set for TCP flow truncation, that is, len_limit_1 and len_limit_2 ($\text{len\_limit}_1 > \text{len\_limit}_2$). They correspond to two situations. The first situation is that the TCP payload lengths of all packets in the entire TCP flow are zero. In this case, the strategy we adopt is to truncate the TCP flow according to len_limit_1. Then, a sequence of length len_limit_1 is obtained. All elements in this sequence are 0. The second situation is that the number of packets with payload in the TCP flow exceeds len_limit_2. In this case, the flow is truncated at the position $P(\textit{len}\_\text{limit}_2)$. $P(x)$ is a function to obtain the position (index) of the x-th packet with payload in the flow. Hence, $P(\textit{len}\_\text{limit}_2)$ can return the index of the $\textit{len}\_\text{limit}_2$-th packet with payload in the flow. $\textit{len}\_\text{limit}_1$ has a higher priority than $\textit{len}\_\text{limit}_2$. Therefore, if $P(\textit{len}\_\text{limit}_2) > \textit{len}\_\text{limit}_1$, truncation is performed according to $\textit{len}\_\text{limit}_1$. If the number of packets in a complete TCP flow does not exceed $P(\textit{len}\_\text{limit}_2)$ and $\textit{len}\_\text{limit}_1$, all packets are reserved. In addition, the TCP flags are used to determine the beginning and end of the flow. The SYN flag indicates that a new TCP flow has started. If there is no SYN packet in a TCP flow, the flow can be considered incomplete. In this paper, the incomplete flows will be directly discarded. The FIN flag and RST flag indicate the end of a TCP flow.

3.4. Flow Preprocessing. The flow preprocessing module is responsible for preliminarily segmenting the collected flows in a window according to the IP address and TCP port numbers. In this way, it can determine which flows have communication relations (as (2)) and which flows have the same service. The flow preprocessing module consists of three parts, namely, IP Partition, Port Partition, and Flow First Time Filter.

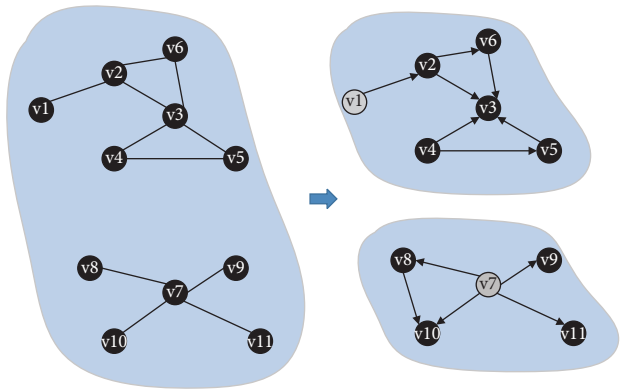

Figure 2: The hosts with communication relationships are separated based on the IP adjacency graph. The gray dots {*v*1, *v*7} are the edges of the subgraphs. By traversing the graph from the boundary nodes, the subgraph can be obtained.

3.4.1. IP Partition. The IP addresses of the flows captured by network traffic acquirer are regarded as nodes. If there is a TCP flow between two IP addresses, an edge is connected between the nodes corresponding to the two IP addresses. In this way, an undirected graph G is constructed to represent the connection relationship between hosts, as shown in the left subgraph of Figure 2. The undirected graph G can be represented algebraically by the adjacency matrix. Firstly, the source IP addresses and destination IP addresses of all the flows are extracted. Then, the duplicate IP addresses are removed. Finally, we construct the adjacency matrix M corresponding to the undirected graph G according to whether there are TCP flows between these IP addresses. The adjacency matrix M is a square matrix. The size of M is the number of unique IP addresses in the TCP flow collection. If there are TCP flows between ip_i and ip_j, the elements at the

```
Require: M, v
Ensure: Block_v
  if v ∉ Block_v then
    Block_v = Block_v ∪ v
  else
    return
  end if
  for v_s ∈ v_all do
    if M[v, v_s] == 1and v_s ∉ Block_v then
      Recursively call FindBlocks to add all vertices adjacent to v_s to the Block_v
    end if
  end for
  return trunc_packetlen_seq[0: packet_index]
```

Algorithm 2: FindBlocks, the algorithm for finding the blocks of nodes.

positions (i, j) and (j, i) in the adjacency matrix M are set to 1. Otherwise, the elements are set to 0. Therefore, the adjacency matrix is symmetric about the main diagonal.

In a flow collection, there are local connections to form a subgraph structure, which represents the block of nodes. For example, the left of Figure 2 contains two subgraphs. Each subgraph in G needs to be analyzed separately. IP Partition can divide the hosts into blocks according to the connection relationship. The schematic diagram of IP Partition is shown in Figure 2. In Figure 2, there are two blocks of nodes, namely, $\text{Block}_1 = \{v1, v2, v3, v4, v5, v6\}$ and $\text{Block}_2 = \{v7, v8, v9, v10\}$. The hosts in Block_1 are connected by edges with each other. The same is true between the hosts in Block_2. However, Block_1 and Block_2 are independent of each other. There is no connection relationship between the hosts in Block_1 and the hosts in Block_2. To extract the blocks from G, two steps must be performed. Firstly, the boundary node of the block needs to be located. The boundary nodes only have adjacent edges to nodes in the block where they are located. Then, all nodes in the block can be obtained by walking through the graph from different boundary nodes. If there is an edge connecting two vertices, it can walk from one vertex to another. The algorithm for finding the nodes of the same block is shown in Algorithm 2.

In Algorithm 2, M is the adjacency matrix, and v is a vertex of M. Block_v denotes the block to which v belongs. *v_all* represents the set of vertices in M.

To find the "boundary" nodes, the undirected graph G needs to be transformed into a directed graph D through orientation. Due to the bidirectional nature of TCP flows, we use arbitrary orientation in this paper. Firstly, the nodes in the undirected graph G are assigned consecutive numbers. Assuming that there are n nodes in graph G, the numbers of these nodes are one to n. Then, the direction of all edges in graph G is determined from the node with the smaller number to the node with the larger number. In this way, the undirected graph G is converted into directed graph D, that is, $G \longrightarrow D$.

Let M_G and M_D denote the adjacency matrix of undirected graph G and directed graph D, respectively. According to the orientation process, it can be concluded that $M_D = UpTriu(M_G)$. $UpTriu(M_G)$ is the upper triangular matrix of M_G. The in-degree and out-degree of the vertex can be calculated by M_D. There is exactly one directed edge between two vertices in the directed graph D. Therefore, if the in-degree or the out-degree of the vertex is zero, the vertex is located in the "boundary" of the subgraph. Formally, V^- refers to the set of vertices whose in-degrees are 0, and V^+ refers to the set of vertices whose out-degrees are 0, as defined in

$$\begin{aligned} V^- &= \{v \mid v \in G, \text{and } d^-(v) = 0\} \\ V^+ &= \{v \mid v \in G, \text{and } d^+(v) = 0\}. \end{aligned} \tag{4}$$

Vertices in both V^- and V^+ can be used to determine the boundary vertices. In this paper, the vertices in V^- are used to find boundary vertices. As shown in the subfigure on the right side of Figure 2, V^- contains two vertices, namely, $v1, v7$. When starting from the vertices of V^- and walking through the undirected graph G, the subgraphs (blocks) are obtained. The algorithm for dividing all nodes into different blocks according to the connection relationship is shown in Algorithm 3. In Algorithm 3, Sections is the set of all blocks.

3.4.2. Port Partition. In the above sections, we have divided different nodes into different blocks according to the communication relationship between nodes. In this section, the Port Partition module aggregates TCP flows between hosts in the same block.

Given an IP address ip_i in a block, according to (3), we can get the set FC of TCP flows that have communication relationships. As introduced in Section 3.2, the Port Partition module firstly divides FC into attack flows and infection flows and then analyzes them from two perspectives of the bot and the vulnerable victim. The attack flows have the following two characteristics when they are observed from the perspective of the bot: (i) the destination port numbers of all attack flows are the same, and (ii) the initiator of the TCP flows is the bot. In addition, there are different characteristics when observing the attack flows from the perspective of the vulnerable victim: (i) The source port numbers are the same, and (ii) the initiator of the TCP flows is the bot. As shown in Figures 3 and 4, the direction of the arrow is from the initiator of the TCP stream to the receiver. Figure 3 shows the attack flows from the perspective of the bot. The IP shared by these TCP flows is the

```
Require: M # adjacency matrix of undirected graph G
Ensure: Sections M_D = UpTriu(M)
  for column of M_D do
    # get the set of in − degrees of all vertices
    InDegree.append(sum (column))
  end for
  for v ∈ M do
    # get the boundary vertices
    if InDegree(v) == 0 then
      boundary_vertices.append (v)
    end if
  end for
  # divide all nodes into blocks
  for v ∈ boundary_vertices do
    block_v = FindBlocks(M, v)
    Sections.append (block_v)
  end for
```

Algorithm 3: FindSections, dividing all nodes into blocks.

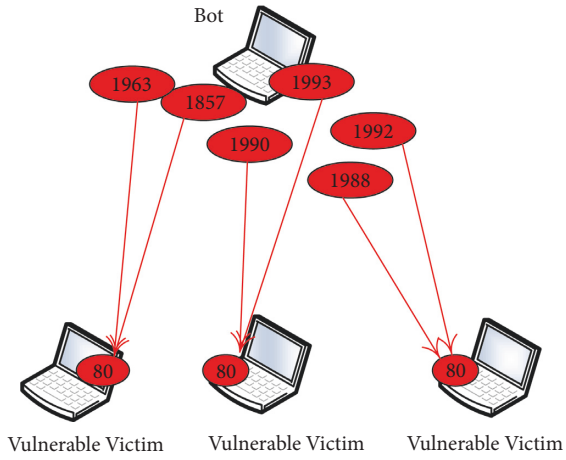

Figure 3: Attack flows from the perspective of bot.

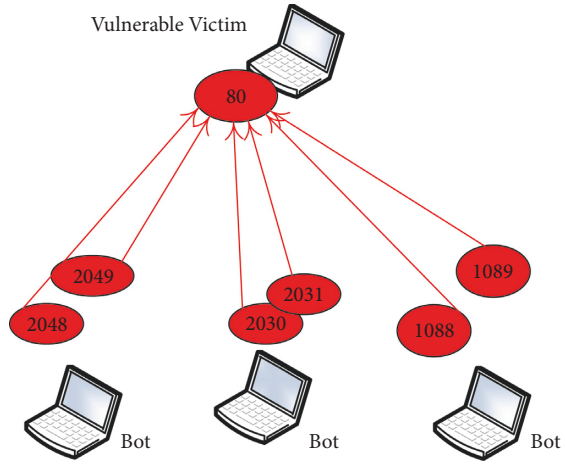

Figure 4: Attack flows from the perspective of vulnerable victim.

IP of the bot. Moreover, the attack flows from the perspective of the vulnerable victim are shown in Figure 4. The hosts at the noncentral location of these TCP streams are bots. There is one bot shown in Figure 3, and there are three bots shown in Figure 4.

However, the features of infection flows are different. When observing them from the perspective of the bot, there are the following two characteristics: (i) the infection flows have the same source port number, and (ii) the initiator of the TCP flows is the bot. In addition, when observing the infection flows from the perspective of the vulnerable victim, there are the following characteristics: the infection flows have the same destination port number, and the initiator of the TCP flows is the bot. No matter from which point of view, the initiators of the TCP flows are always the bots, as shown in Figures 5 and 6.

Based on the above analysis, we get the following port division schemes. Firstly, the directions of TCP flows in the set FC are adjusted to take ip_i as the "source direction." Then, these flows are clustered according to the following four strategies: (i) the flows that with the same destination port number and whose initiators are ip_i, (ii) the flows that with the same source port number and whose initiators are not ip_i, (iii) the flows that with the same source port number and whose initiators are ip_i, and (iv) the flows that with the same destination port number and whose initiators are not ip_i. The attack flows are aggregated based on (i) and (ii). The infection flows are aggregated based on (iii) and (iv). The initiator of the flows is determined by the Flow First Time Filter.

3.4.3. Flow First Time Filter. Flow First Time Filter module determines the initiator of the TCP flows according to the timestamp of the first packet of the TCP flows. Given an IP address ip_i, the TCP flow with ip_i as the source address is downstream, and the TCP flow with ip_i as the destination address is upstream. If the timestamp of the first packet of the downstream flow is less than that of the first packet of the upstream flow, the initiator of this TCP flow is ip_i; otherwise, the initiator is not ip_i.

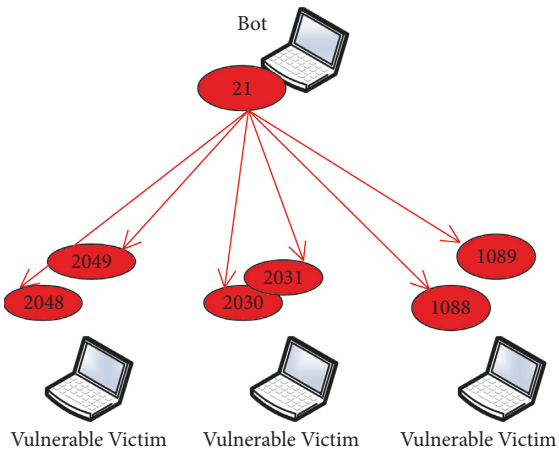

Figure 5: Infection flows from the perspective of bot.

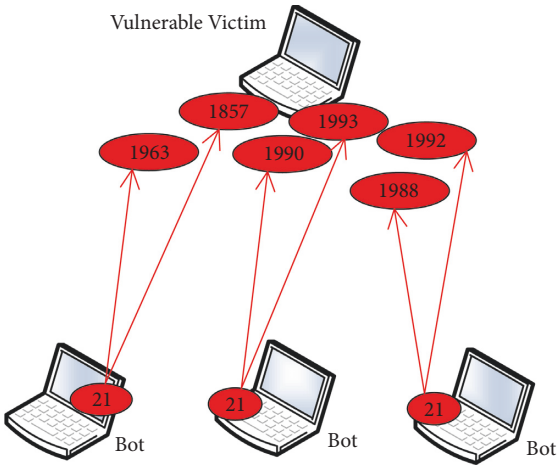

Figure 6: Infection flows from the perspective of vulnerable victim.

3.5. Malicious Flow Recognition. The group nature of botnet makes that the flows of bots often present a certain similarity between the flows of the bots. Once the botnet is active, the traffic generated by different bots has a high similarity with each other. In addition, some botnets use encryption algorithms to avoid detection. However, the relationship between the length of packets in a flow will not be affected by the encryption algorithm. In this paper, we focus on the method of calculating the similarity of TCP flows. The sequence of the packet length is adopted to evaluate the similarity of the flows. The method we adopt to calculate the similarity of the packet length sequence of TCP flows is the Levenshtein algorithm [27]. If the two sequences are completely the same, the similarity is 1. If the two sequences are completely different, the similarity is 0. The Levenshtein algorithm is mainly used to calculate the distance between two strings, which is the minimum number of editing operations required to convert one string to another. Editing operations allowed during the conversion process include (i) replacing one character with another character, (ii) inserting a character, and (iii) deleting a character.

Given two strings a and b, the Levenshtein algorithm can be formally defined as (5) to calculate the similarity between the string a and b. In (5), $i\,(i > 0)$ represents the i-th position of string a, and $j\,(j > 0)$ represents the j-th position of string b. When $i = 0$ or $j = 0$, the distance of string a and b is zero. Let SimRatio denote the similarity of the packet length sequences. Then, $\text{SimRatio}(a, b)$ equals $lev_{a,b}(len(a), len(b))$, where $len(a)$ is the length of string a.

$$lev_{a,b}(i, j) = \min \begin{cases} lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + 1_{a_i \neq b_j}. \end{cases} \quad (5)$$

When calculating the similarity of the sequences of the packet length, the sequences of packet length are regarded as strings. Hence, the Levenshtein algorithm is applicable. In actual application, some tips are introduced to improve the performance of the algorithm. Firstly, suppose that the elements of the two sequences are the same, and only the length of the two sequences is compared. The similarity of two sequences in terms of length Sim_{len} is defined as (6). Given the similarity threshold Sim_{thre}, if Sim_{len} is less than the similarity threshold Sim_{thre}, it can be directly recognized that the two sequences are different, and the Levenshtein algorithm is no longer required. The complexity of Sim_{len} is much less than that of the Levenshtein algorithm. Therefore, the computational complexity can be reduced when calculating the similarity of sequences.

$$\text{Sim}_{len} = 1 - \frac{|len(s_1) - len(s_2)|}{len(s_1) + len(s_2)}. \quad (6)$$

The overall process of bot detection is shown in Figure 7. In Figure 7, the dots are used to represent the hosts. The gray dots represent the hosts in the internal network, and the black dots represent the hosts in the external network. The black solid lines indicate that there are TCP flows between the hosts. The dotted lines represent TCP flows. Figure 7 shows an IP Block with 10 hosts, namely, $n_1, n_2, \ldots, n_{10}$. We analyze the hosts in the IP Block in turn. The process of analyzing the host n_1 is shown in the dashed box. First, the TCP flows that have a communication relationship with the host n_1 are collected. These TCP flows are denoted as $FC = \{f_1, f_2, f_3, \ldots, f_{12}\}$. Then, the flows in FC are divided by the Port Partition algorithm to generate some TCP flow blocks. Finally, the Levenshtein algorithm is used to calculate the similarity of these flows in the flow blocks. The flows with high similarity (exceeding the threshold Sim_{thre}) are regarded as malicious flows, and the bots are identified according to the strategy adopted in the Port Partition algorithm. In Figure 7, $\{f_1, f_4\}, \{f_{10}, f_{12}\}$ are finally detected as malicious flows. Therefore, the host n_1 can be identified as a malicious bot.

4. Experimental Analysis

The detection performance of the proposed bot detection method is evaluated in this paper. The detection performance is mainly evaluated from three aspects: (i) the

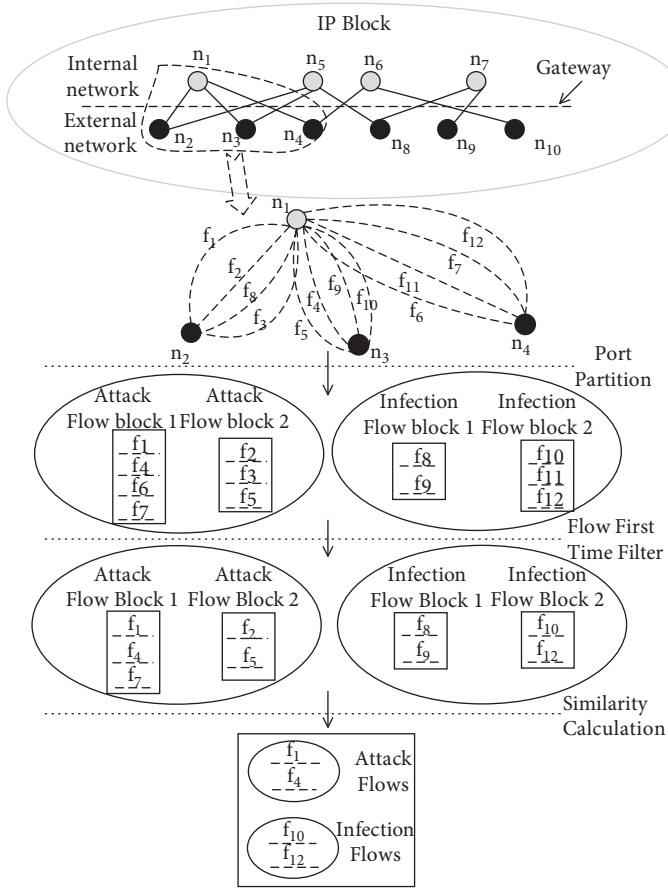

Figure 7: The overall process of bot detection.

detection accuracy (true positive rate and false positive rate) of the detection system on the dataset ISCX, (ii) the influence of different parameter settings on the detection effect, and (iii) comparing the performance of the method proposed in [13].

4.1. Dataset. We employ the ISCX botnet dataset for our experiments. ISCX [12] is a publicly available dataset that combines nonoverlapping subsets of three other datasets. ISCX dataset contains traffic from 16 different IRC, P2P-based, and HTTP-based botnets, which makes the ISCX dataset more general, realistic, and representative. The ISCX dataset consists of two subsets: training and testing. The training dataset is 5.3 GB in size, of which 43.92% is malicious traffic (including 7 types of botnets). The testing dataset is 8.5 GB in size, of which 44.97% is malicious traffic (including 16 types of botnets). Figures 8 and 9 show the distribution of the number of TCP flows in the ISCX dataset. Figure 8 shows the traffic distribution in the training dataset, and Figure 9 is about the testing dataset.

4.2. Experimental Evaluation. Since the detection method we designed does not require a training process, the training dataset and the testing dataset are treated the same, and we conducted experimental evaluations in both datasets. The experimental results are compared with [13] from two aspects: bot detection rate (TPR) and false alarm rate (FPR), as defined in

$$TPR = \frac{\text{the number of detected bots}}{\text{total number of bots}},$$
$$FPR = \frac{\text{false alerts}}{\text{total number of benign hosts}}. \quad (7)$$

TPR measures whether our method can effectively detect bots. FPR measures the side effect of our method that benign hosts are incorrectly identified as bots. The higher TPR is, the better it is. The lower FPR is, the better it is.

There are two parameters that need to be set. One is the size of the flow window WinSize. The other is the similarity threshold Sim_{thre}. Firstly, we set the similarity threshold $\text{Sim}_{thre} = 0.99$ to observe the influence of different flow window sizes on the recognition results. WinSize is set to 10, 50, 100, and 200 in turn, and the recognition results are recorded for comparison, as shown in Table 1. When the WinSize increases, the method proposed in this paper can detect traffic in a larger range, which helps to improve the detection rate of the method. In addition, the number of benign traffic flows will also increase with the increase of WinSize, and the probability of detection errors will also

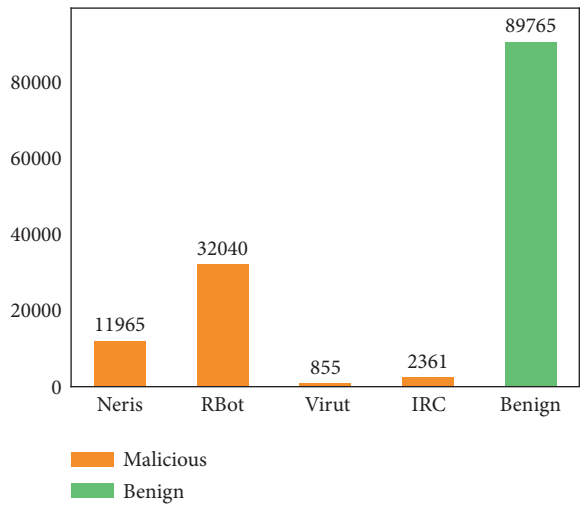

Figure 8: TCP flow distribution in the training dataset. The abscissa represents the botnet, and the ordinate represents the number of flows.

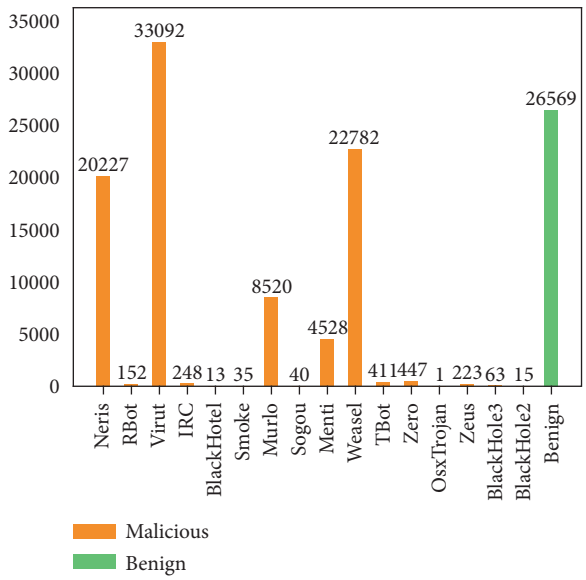

Figure 9: TCP flow distribution in the testing dataset. The abscissa represents the botnet, and the ordinate represents the number of flows.

increase slightly. Therefore, *FPR* increases to a certain extent with the increase of WinSize.

The experimental results show that a high detection rate can be achieved without setting an excessively large window size. When the window size continues to increase, *TPR* will quickly reach the optimal stable state. However, *FPR* increases slightly. The experimental results show that a large window size will increase the false positive rate. In addition, there are 28 bot hosts (or host pairs) in the ISCX-Testing dataset. Our method successfully detects 27 of them (27/28 = 0.96428); only *Osx*_trojan (IP: 172.29.0.109) is not detected. The reason is that there is only one *Osx*_trojan TCP flow in the ISCX-Testing dataset, as shown in Figure 9. Our detection method requires at least two related TCP flows to get a conclusion. Hence, the detection of *Osx_trojan* failed.

Table 1: Evaluate the effect of different WinSize on the results.

WinSize	ISCX-training dataset		ISCX-testing dataset	
	TPR	FPR	TPR	FPR
10	1.0	0.00706	0.92857	0.05034
50	1.0	0.00797	0.96428	0.06013
100	1.0	0.00813	0.96428	0.07015
200	1.0	0.00889	0.96428	0.08041

In addition, we set the flow window size WinSize = 10 to observe the influence of different thresholds Sim_{thre} on the recognition results. Sim_{thre} is set to 0.7, 0.8, 0.9, and 0.99 in turn, and then the recognition results are recorded for comparison. The results are shown in Table 2.

The experimental results show that the optimal effect can be achieved in the training dataset when Sim_{thre} is set to 0.7. With the increase of Sim_{thre}, the detection rate and false alarm rate have not changed. In the testing dataset, the detection rate does not change when Sim_{thre} keeps increasing, but the false alarm rate (*FPR*) gradually decreases. Therefore, the larger Sim_{thre}, the lower the false alarm rate.

Many research works choose different datasets for method verification. The verification results are different by selecting different datasets. Therefore, to be relatively fair, we compare the methods proposed in this paper with those of others who also choose the ISCX dataset to verify the model effects. The methods in Table 3 have achieved remarkable results in botnet detection. Meanwhile, they are influential research works. The authors of [13] propose an adaptive botnet detection framework, which uses the SVM model to detect botnet. They train the model on the ISCX-training dataset and then evaluate the effect on the ISCX test dataset. Beigi et al. [8] focus on the proper selection and experimental assessment of features for accurate detection of botnets. Mohammad Alauthaman et al. [28] present a method based on an adaptive multilayer feedforward neural network in cooperation with decision trees to detect P2P-based bots. Soodeh Hosseini et al. [22] use a novel botnet detection and classification method based on convolution neural networks and negative selection algorithms. They all more or less select the ISCX dataset or partial samples in the dataset to verify the performance of the proposed methods. The comparison results are shown in Table 3. Through the comparison of the experimental results, it can be seen that our method is more effective.

4.3. Flow Window Fluctuation. Since WinSize is the minimum value of the flow window, the size of the flow window fluctuates actually, as shown in Figure 10.

The fluctuation of the flow window size affects the use of memory, which is an important aspect of model performance. Figure 10 shows the fluctuation of the size of the flow window when setting different WinSize. Figures 10(a) and 10(b) are the fluctuations of the flow window size in the

Table 2: Evaluate the effect of different Sim_{thre} on the results.

Sim_{thre}	ISCX-training dataset		ISCX-testing dataset	
	TPR	FPR	TPR	FPR
0.7	1.0	0.00706	0.92857	0.05239
0.8	1.0	0.00706	0.92857	0.05193
0.9	1.0	0.00706	0.92857	0.05056
0.99	1.0	0.00706	0.92857	0.05034

Table 3: Comparison of experimental results.

Methods	TPR	FPR
Javier et al. [13]-1	1.0	0.082
Javier et al. [13]-2	0.48	0.0017
Beigi et al. [8]	0.75	0.023
Alauthaman et al. [28]	0.992	0.0075
Hosseini et al. [22]	0.99	—
Our method	**1.0**	**0.00706**

"—" indicates that the corresponding result is not presented in the paper.

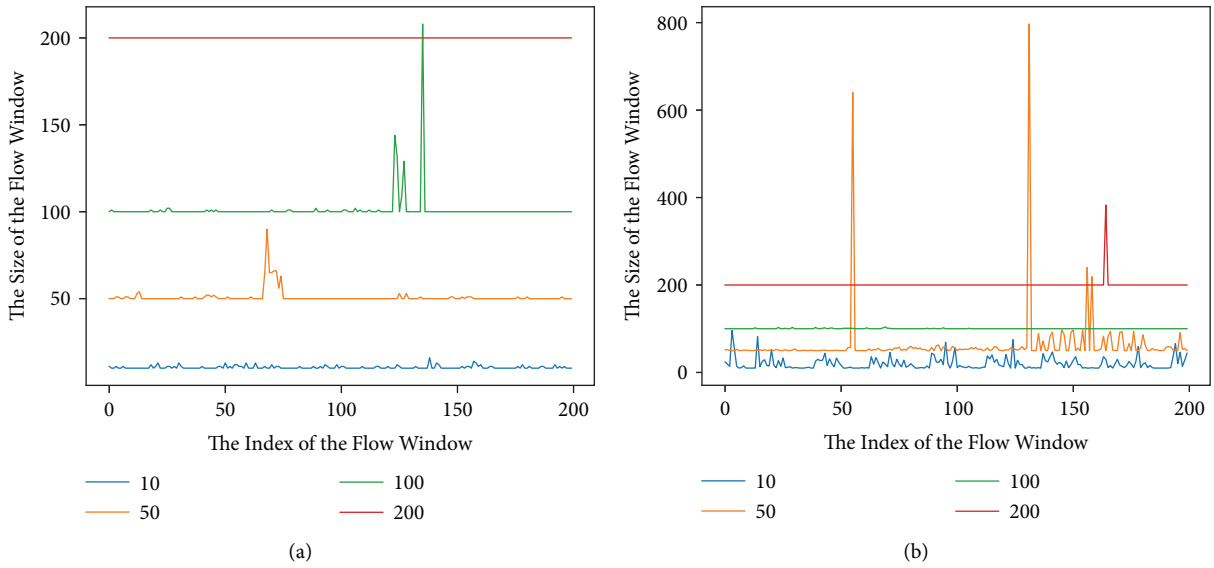

Figure 10: The fluctuation of the flow window size with different WinSize. (a) and (b) are the fluctuations of the flow window size in the training and testing datasets, respectively.

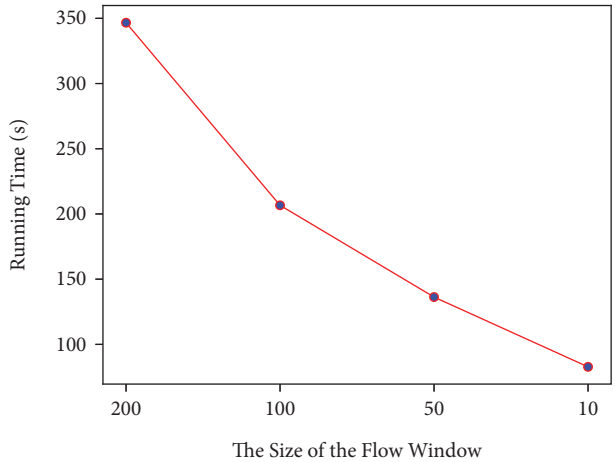

Figure 11: The impact of different WinSize on runtime.

training and testing datasets, respectively. It can be seen that most of the window sizes fluctuate around the WinSize, except for a sharp increase in the size of individual windows.

In addition, the running time of our detection method implemented in python is evaluated on a personal laptop (Intel i7-6500U CPU, 2.59 GHz, 16 GB Memory, Windows 10) with the ISCX-Testing dataset. When evaluating the running time, the time of the network traffic acquirer module is not considered. TCP flow windows are continuously fed to the subsequent TCP flow processing modules (preprocess module, attack flow recognizer module, infection flow recognizer module, and result integration module). The running time is shown in Figure 11. The result is that the larger the window, the longer the running time.

5. Conclusion

In this paper, we have proposed a protocol-independent bot detection framework based on the similarity of flows to detect botnets. The proposed method does not rely on the protocol and structure of botnets, which exploits the fact that all botnets have group characteristics and the sequence of packet length is not affected by encryption. Therefore, the sequence of packet length is used as the characteristic of the TCP flow, and the similarity of TCP flows is calculated to detect botnet traffic. We evaluated the experimental results on the ISCX dataset, and the results show that our method has excellent performance.

In the future, we will consider UDP packets to better deal with the new botnet technology. Meanwhile, we will make the detection system more robust and prevent botnets from using UDP to escape detection. In addition, the performance of the system will be further optimized to enable the system to process traffic in real-time.

Data Availability

Complete information about datasets is available at https://iscx.ca/botnet-dataset.

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 (Grant no. 62002374).

References

[1] H. K. Idriss, "Mirai botnet in Lebanon," in *Proceedings of the 2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6, Beirut, Lebanon, June, 2020.

[2] N. Kaur and M. Singh, "Botnet and botnet detection techniques in cyber realm," in *Proceedings of the 2016 International Conference on Inventive Computation Technologies (ICICT)*, pp. 1–7, IEEE, Coimbatore, India, August, 2016.

[3] S. A. Yeshwantrao and V. J. Jadhav, "Threats of botnet to internet security and respective defense strategies," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 1–7, 2014.

[4] P. Ping Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 2, pp. 113–127, 2010.

[5] M. Cirillo, M. D. Mauro, V. Matta, and M. Tambasco, "Botnet identification in ddos attacks with multiple emulation dictionaries," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3554–3569, 2021.

[6] M. Fazil, A. K. Sah, and M. Abulaish, "Deepsbd: a deep neural network model with attention mechanism for socialbot detection," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4211–4223, 2021.

[7] W. Wang, B. Fang, Z. Zhang, and C. Li, "A novel approach to detect irc-based botnets,"vol. 1, pp. 408–411, in *Proceedings of the 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 1, IEEE, Wuhan, China, April, 2009.

[8] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proceedings of the 2014 IEEE Conference on Communications and Network Security*, pp. 247–255, IEEE, San Francisco, CA, USA, October, 2014.

[9] R. S. Rawat, E. S. Pilli, and R. C. Joshi, "Survey of peer-to-peer botnets and detection frameworks," *IJ Network Security*, vol. 20, no. 3, pp. 547–557, 2018.

[10] N. Wolley and C. Wolley, "Botsuer: suing stealthy p2p bots in network traffic through netflow analysis," *Cryptology and Network Security*, pp. 162–178, 2013.

[11] G. Gu, R. Perdisci, J. Zhang, W. Lee, and Botminer, "Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium*, pp. 139–154, Quebec, Canada, August, 2008.

[12] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[13] J. Álvarez Cid-Fuentes, C. Szabo, and K. Falkner, "An adaptive framework for the detection of novel botnets," *Computers & Security*, vol. 79, pp. 148–161, 2018.

[14] T. S. Wang, H. T. Lin, W. T. Cheng, and C. Y. Chen, "Dbod: clustering and detecting dga-based botnets using dns traffic analysis," *Computers & Security*, vol. 64, pp. 1–15, 2017.

[15] J. Paschalidis, I. C. Paschalidis, Botnet detection based on anomaly and community detection," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, 2017.

[16] J. Kwon, J. Lee, H. Lee, and A. Perrig, "Psybog: a scalable botnet detection method for large-scale dns traffic," *Computer Networks*, vol. 97, pp. 48–73, 2016.

[17] D. Chang and J. M. Chang, "Enhanced peerhunter: detecting peer-to-peer botnets through network-flow level community behavior analysis," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1485–1500, 2019.

[18] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, "Using machine learning technliques to identify botnet traffic," in *Proceedings of the 2006 31st IEEE Conference on Local Computer Networks*, pp. 967–974, IEEE, Tampa, FL, USA, November, 2006.

[19] M. Alauthman, N. Aslam, M. Al-kasassbeh, S. Khan, A. Al-Qerem, and K. K. Raymond Choo, "An efficient reinforcement learning-based botnet detection approach," *Journal of Network and Computer Applications*, vol. 150, Article ID 102479, 2020.

[20] A. Acarman and T. Acarman, "Deep learning to detect botnet via network flow summaries," *Neural Computing & Applications*, vol. 31, no. 11, pp. 8021–8033, 2019.

[21] S. Mousavi, M. Khansari, and R. Rahmani, "A fully scalable big data framework for botnet detection based on network traffic analysis," *Information Sciences*, vol. 512, pp. 629–640, 2020.

[22] S. Hosseini, A. E. Nezhad, and H. Seilani, "Botnet detection using negative selection algorithm, convolution neural network and classification methods," *Evolving Systems*, vol. 13, no. 1, pp. 101–115, 2022.

[23] A. Alsubhi and K. Alsubhi, "Botnet detection approach using graph-based machine learning," *IEEE Access*, vol. 9, Article ID 99166, 2021.

[24] R. Biswas and S. Roy, "Botnet traffic identification using neural networks," *Multimedia Tools and Applications*, vol. 80, no. 16, p. 24, 2021.

[25] J. Velasco-Mata, V. Gonzalez-Castro, E. F. Fernandez, and E. Alegre, "Efficient detection of botnet traffic by features selection and decision trees," *IEEE Access*, vol. 9, Article ID 120567, 2021.

[26] W. N. H. Ibrahim, S. Anuar, A. Selamat et al., "Multilayer framework for botnet detection using machine learning algorithms," *IEEE Access*, vol. 9, Article ID 48753, 2021.

[27] R. Myers, R. Wison, and E. R. Hancock, "Bayesian graph edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 628–635, 2000.

[28] M. Alauthaman, N. Aslam, L. Zhang, R. Alasem, and M. A. Hossain, "A p2p botnet detection scheme based on decision tree and adaptive multilayer neural networks," *Neural Computing & Applications*, vol. 29, no. 11, pp. 991–1004, 2018.