



Article

Fine-Grained High-Utility Dynamic Fingerprinting Extraction for Network Traffic Analysis

Xueying Sun ¹, Junkai Yi ^{1,2,*}, Fei Yang ¹ and Lin Liu ³¹ School of Automation, Beijing Information Science and Technology University, Beijing 100192, China² Key Laboratory of Modern Measurement and Control Technology, Ministry of Education, Beijing 100192, China³ China Information Technology Security Evaluation Center, Beijing 100085, China

* Correspondence: yijk@bistu.edu.cn

Abstract: Previous network feature extraction methods used for network anomaly detection have some problems, such as being unable to extract features from the original network traffic, or that they can only extract coarse-grained features, as well as that they are highly dependent on manual analysis. To solve these problems, this paper proposes a fine-grained and highly practical dynamic application fingerprint extraction method. By putting forward a fine-grained high-utility dynamic fingerprinting (Huf) algorithm to build a Huf-Tree based on the N -gram (every substring of a larger string, of a fixed length n) model, combining it with the network traffic segment-IP address transition (IAT) method to achieve dynamic application fingerprint extraction, and through the utility of fingerprint, the calculation was performed to obtain a more valuable fingerprint, to achieve fine-grained and efficient flow characteristic extraction, and to solve the problem of this method being highly dependent on manual analysis. The experimental results show that the Huf algorithm can realize the dynamic application of fingerprint extraction and solve the existing problems.



Citation: Sun, X.; Yi, J.; Yang, F.; Liu, L. Fine-Grained High-Utility Dynamic Fingerprinting Extraction for Network Traffic Analysis. *Appl. Sci.* **2022**, *12*, 11585. <https://doi.org/10.3390/app122211585>

Academic Editors: Guang Cheng, Christos Bouras, Shui Yu and Yongning Tang

Received: 8 October 2022

Accepted: 11 November 2022

Published: 15 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: application fingerprinting; N -gram model; fine-grained analysis; flow processing; feature extraction

1. Introduction

Anomaly detection is a very important network security method. It can detect anomalies by extracting network features, establishing a behavior standard model, and calculating sample offset. In the real network environment, huge raw traffic will be generated, which contains a large number of data packets. These data packets often contain a lot of valuable information and can be analyzed by what the operations terminal is doing. However, most of the previous anomaly detection methods can only analyze the features generated by specific systems and cannot directly extract the features from the original network traffic.

The feature that can uniquely mark network behavior and identify a certain network behavior is defined as the application fingerprint [1]. The fingerprint is a relatively broad concept in network security. Any feature that is different from other applications and can identify applications can be defined as a fingerprint. It is necessary to use application fingerprint to identify smartphone applications. Taylor et al. [2] use application fingerprint to identify some information of smartphone applications.

In [3–11], it can be found that the application fingerprint can be implemented in various ways, and its purpose is to extract features to uniquely mark the communication behavior of an application. The target object of fingerprint extraction can be TCP messages, HTTPS encrypted packets, log events, Application Protocol data units, and others. Elements in a fingerprint can consist of message types, statistical data, or others. Application fingerprints can be expressed by sequences, vectors, state graphs, and logical expressions. The extracted application fingerprints are mainly used to process encrypted traffic, mostly for traffic

classification and anomaly detection, and for network security problems such as application identification and user behavior analysis.

In addition, it can be found from the literature that most of these application fingerprints are generated by manual analysis, and a few are generated by the clustering algorithm. This reflects the high dependence on manual analysis in actual fingerprinting extraction.

Some relevant literature in the recent years [12–16] combines network behavior analysis with natural language processing. The basic idea is to understand network communication as a knowledge exchange between client and server, similar to language communication between people, so the packet of communication can be regarded as a “language” of computer communication. Extraction and communication on both sides of the “language” need to be conducted in a certain way, and normally this kind of communication is the process of scheduling, which is produced following the time of data. Therefore, the most direct representation is a sequence or a sequence of elements in the sequence, so the order is also the kind of characteristics of elements, and in the sequence characteristics of said method, each item represents different information, such as packet size, packet payload, user sessions of logs, messages, packet number, etc. Although the sequence considers the order of each element, it does not represent the dependencies between items, so to understand the communication process from the perspective of natural language processing, it is necessary to represent the relations between the elements before and after.

The N -gram (every substring of a larger string, of a fixed length n) model [17] is a common representation model in natural language processing. The N -gram model analyzes from the perspective of semantics, where words are interdependent of each other, and the occurrence of the current word is related to the previous N words and has nothing to do with the previous $N + 1$ words, which is the meaning of the N -gram model. The most important thing in the N -gram model is the relationship between the probability of the current word and the first N words. Therefore, the expression form can be similar to a sequence, state graph, or tree, which will be introduced in the following paragraphs. Duessel et al. [18] adopted the N -gram model with HTTP Requests as elements in the model and represented the N -gram model in the form of a tree. Wang et al. [19] also adopted the N -gram model to express network behavior and also used HTTP Requests as elements to express the N -gram model in the form of sequence.

Therefore, using the N -gram model in natural language processing can more accurately describe the process of communication between the two ends, especially taking the message type as the element of the model.

The characteristics of network behavior analysis in recent years are from coarse grained to fine grained. Coarse grained and fine grained are relative concepts. They are distinguished according to the level of detail in the division of project modules, said to be fine-grained. Relative to fine grained, the larger each module (or sub-module) is, the coarser the work it is responsible for, which is called coarse grained. Some methods can directly analyze raw network traffic but can only extract coarse-grained features, not fine-grained features of a single mobile terminal. Moreover, anomaly detection can only rely on manual analysis methods of static datasets to obtain features, which is not suitable for dynamic and real network environments with huge data traffic. According to the existing problems, this paper proposes the fine-grained high-utility dynamic fingerprinting (Huf) algorithm to focus on solving the above problems.

Given the existing problems, this paper proposes an approach to the fine-grained high-utility dynamic application fingerprinting extraction to solve them. In this paper, the original network traffic is taken as the analysis target, and the characteristics of the original network traffic are extracted by analyzing the communication session—a fingerprint is applied to solve the problem that the original network traffic cannot be directly dealt with by anomaly detection, and the fine-grained anomaly detection can be realized by a fingerprint. Then, the frequent item set mining algorithm is improved to realize the automatic extraction of network features and solve the problem that feature extraction depends on manual analysis.

Extracting the characteristics of network traffic has always been the focus of network behavior analysis. Modeling a large number of independent data packets is the premise of traffic processing. The Huf algorithm proposed in this paper can realize dynamic application fingerprint extraction, solve the problem that anomaly detection cannot process the original network dataset and solve two problems that rely on manual feature extraction, and implement dynamic data stream processing.

This paper consists of six sections: Section 1 is an introduction; Section 2 is a description of the problem, including the proposal and innovation of the problem and the process of the proposed method; Section 3 is the preprocessing of network traffic in the new method proposed in this paper; Section 4 is the efficient and dynamic fingerprint extraction method proposed in this paper; Section 5 is the experimental analysis; Section 6 is the conclusion of this paper.

2. Problem Description

The goal of this paper is to automatically extract application behavior features by analyzing network raw traffic directly obtained in the network. The application fingerprint object adopted in this paper is the most basic form of network traffic data packets. Korczynski and Duda [20] proposed stochastic fingerprints for application traffic flows conveyed in Secure Socket Layer/Transport Layer Security (SSL/TLS) sessions, TLS encryption protocol data packets, and other relevant non-encryption protocol data packets Transport Control Protocol/User Data Protocol(TCP/UDP), Hyper Text Transfer Protocol (HTTP), etc. Fingerprint elements are message types that perform different functions under different protocols. The N -gram model of natural language processing is used to describe the application fingerprint. The extraction methods use a high-utility mining method with frequent item set of real-time online access, without artificial analysis and extraction.

To achieve this goal, the following problems need to be solved.

2.1. Division of Network Traffic

It can be seen from the above that the previous features were mainly based on manual analysis. Network traffic is a continuous data flow with a large amount of unstructured data. The necessary step before feature extraction is to divide the data stream into several small fragments. Zhang et al [21] proposed a novel unsupervised approach to deal with unknown applications. Bhatia et al. [22] provides a way to divide network traffic into flow, session, and dialog step by step according to timestamp and IP address. However, the application fingerprint that needs to be extracted is the fine-grained application fingerprint. Only processing according to time interval will lead to incomplete or overlapping fingerprint extraction, so fine-grained application fingerprints cannot be extracted. In addition, a large number of fragments of the application fingerprint in the extraction process will greatly occupy the time and space of algorithm calculation. therefore, the division here is focused on finding "breakpoints". This "breakpoint" is determined not only by time but by other relevant attributes such as IP address, protocol type, etc., so that a complete fine-grained application fingerprint can be extracted.

2.2. Dynamically Process Traffic

Data will not be processed in a static dataset, and all data can be scanned and processed only once to meet the requirements of huge network traffic. In this way, when the network behavior changes, data can be obtained in time. To meet the requirements of dynamic processing, the proposed algorithm uses a tree based on the N -gram model to save candidate sets and then extracts the application fingerprint from the structure of the tree.

2.3. Application Fingerprints with Typical Characteristics Need to Be Extracted

Network communication is conducted according to the corresponding protocol, which means that most application fingerprints are common structures, such as the "three-way

handshake”, which is the way almost all application communication takes place. If the feature is only extracted from the usage frequency, it will be universal and cannot be used for the next traffic classification. Therefore, the unique features that set each app apart from the others were extracted.

2.4. Solutions

In response to the above problems, the solution in this paper is as follows: preprocess the original network traffic obtained directly from the network and focus on finding “breakpoints” when dividing network traffic; this “breakpoint” no longer only depends on time, but it is determined by other corresponding IP addresses, protocol types, and other related attributes so that a complete fine-grained application fingerprint can be extracted. The algorithm proposed in the dynamic processing uses a tree based on the *N*-gram model to save the candidate set, and then extracts the application fingerprint from the structure of this tree. The last step is to extract the unique features of each application that distinguish it from other applications.

The Huf algorithm proposed in this paper can solve the problem. The Huf algorithm is a fine-grained and efficient dynamic fingerprint extraction method. Its structure is shown in Figure 1. It can be seen that the process of extracting application fingerprints can be divided into two parts: network traffic preprocessing and frequent set mining to obtain application fingerprints. This paper performs data preprocessing in Section 3 and performs fine-grained and efficient dynamic fingerprint extraction in Section 4.

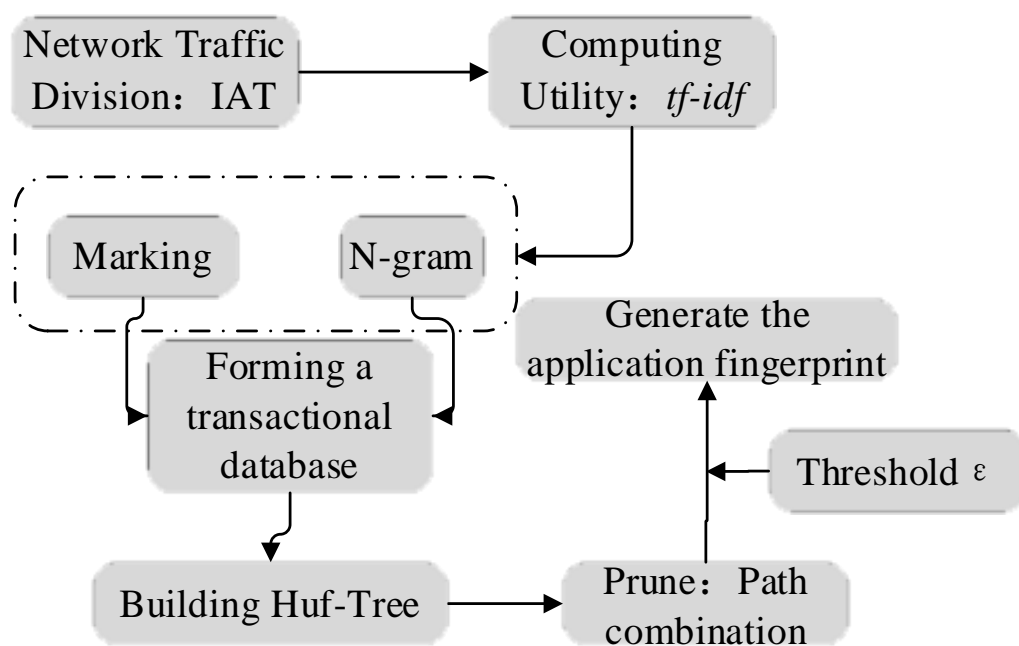


Figure 1. Huf algorithm.

3. Data Processing

It is difficult to deal with unstructured data with a huge amount of data and no typical characteristics. Network traffic is a continuously generated data flow. Before extracting features, the necessary step is to divide the data flow, obtain several small fragments, and further process them. This section first analyzes the relevant content of the TLS Protocol, takes the communication message as the analysis object, marks each message, and then describes the communication process between the client and the server. Then, it analyzes the characteristics of domain names of various application categories, extracts representative domain names, and provides relevant domain name data for exception analysis for further manual analysis of exceptions. In the fourth section, we introduce the process of fine-grained efficient dynamic fingerprint extraction.

Frequent project set mining is an algorithm for discovering relationships between independent items. An improved frequent project set mining algorithm applied to fingerprint extraction is proposed, which has three characteristics: first, the method can process dynamic data, dynamically extract feature fingerprints from continuously generated network data, find the “breakpoint” that divides network traffic, and extract the entire application fingerprint; second, in data analysis, the sequential dependence of messages is the key content of application behavior, and the application fingerprint is established through the N -gram model; third, the utility of the message is used as a screening criterion.

3.1. SSL/TLS Encryption Protocol and Communication Process

This section analyzes TLS1.2, a Transport Layer Protocol that works over TCP. TLS version 1.2 consists of two layers: the TLS Recording Protocol and the TLS Handshake Protocol. The TLS Recording Protocol works under the TLS Handshake Protocol and provides basic functionality. The TLS Logging Protocol defines four specific protocol formats, which are the Handshake Protocol, the Alert Protocol, the Change Password Specification Protocol, and the Application Data Protocol. The upper-layer TLS handshake protocol is primarily used for negotiation sessions.

The TLS Handshake Protocol is superior to the TLS Recording Protocol. When the client and server initiate communication for the first time, the two parties agree on the protocol version, encryption algorithm, digital certificate, etc., and complete the following functions. First is the Hello Message exchange and agreement to carry out algorithms, random numbers, and other information on the server, and check the session recovery. Second, the two sides exchange necessary key parameters, allowing the client and server to directly exchange the Hello Message, the server in the algorithm, the random number, and other information on the agreement, and check the session resumption. Third is the exchange of digital certificates and related parameters so that the client and server can authenticate each other. Fourth, the master key is generated from the pre-master key and random numbers are exchanged with each other. Last, it must be ensured that the client and server can verify each other’s parameters and that the parameters will not be tampered with by attackers.

When the client and server need to use TLS Protocols to establish communication, they must follow the TLS Handshake Protocol to establish communication, which can be divided into three phases.

3.1.1. Establish Communication

The client sends a ClientHello Message to the server. After receiving the request, the server sends a ServerHello Message. Both the ClientHello Message and ServerHello Message contain the information necessary to establish communication.

3.1.2. Exchange Keys and Certificate

The two parties need to exchange keys and certificates. There are four types of messages related to keys: Server Certificates, Server Key Exchanges, Client Certificates, and Client Secret Exchanges. In addition, there is a “Change Password Specification” message and a “Certificate Request” message to complete some functions. Figure 2 describes the process in detail.

3.1.3. Data Transmission

In this phase, the data that the user needs to transfer are transferred in the application data format.

The communication process shown in Figure 2 is the first complete communication process between the client and server. The actual use of encrypted communication is a continuous process. Exchanging keys and certificates for each communication consumes resources and is unnecessary. The server will retain the session ID of communication with the client for a while and wait for the client to continue using the previous session. If

the session ID of the client cannot be found in the existing list of the server, the server initiates a new session and performs a complete handshake. Figure 3 shows a simplified TLS communication process.

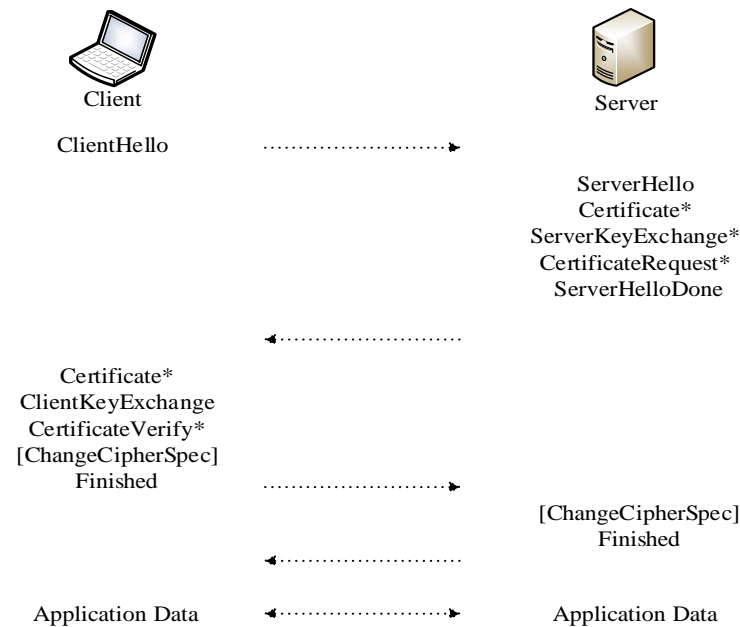


Figure 2. The communication process of Handshake Protocol.

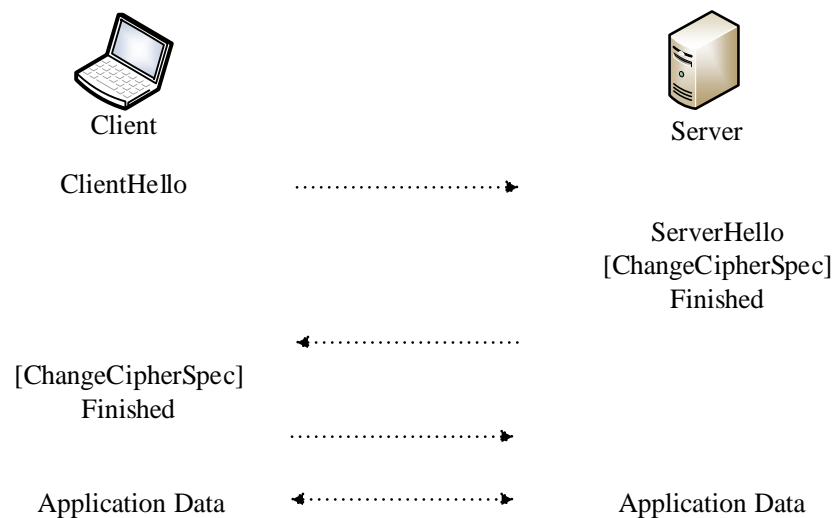


Figure 3. The simplified process of TLS communication.

As shown in Figures 2 and 3, different messages are required at different stages of communication establishment. Each message has a specific format and performs certain functions. In practical applications, programmers generally implement the communication between client and server according to protocol requirements, but also make some adjustments according to function requirements, for example, omitting some unnecessary messages or combining multiple packets to improve communication efficiency. The diversity of messages in communication and the flexibility in practice makes it possible to extract fingerprint features. Table 1 lists all message types and functions of the TLS Handshake Protocol.

The notations in the right-most column of Table 1 are used to describe the communication process. The research object of this paper is encrypted traffic, but in actual network data, common protocols are used to complete some unimportant communication between

client and server. The entire network data is a mixture of encrypted and non-encrypted protocols. Figure 4 shows the hierarchy of common protocols in the network.

Table 1. The messages of TLS Protocol.

Protocol	Message Type	Message Function	Notation
Change Cipher Spec Protocol	ChangeCipherSpec	Both the client and server can send a message to inform each other that the data to be transmitted will be encrypted with the new key.	21:
	Closure Alerts	Notify the other party that no more data will be sent under the current connection.	22:1
Alert Protocol	Error Alerts	The communication parties immediately close the current connection and clear related information, such as the key and session ID.	22:2
	Hello Messages	Hello Request	Sent by the server to inform the client that a new communication negotiation process should be established.
ClientHello		This message is sent when the client needs to establish a connection with the server or as a response to a Hello Request.	23:2
ServerHello		The server responds to the Client Hello packet sent by the client.	23:3
Hello Extensions		When the client has an extension type, the Hello Extensions will appear in response to ClientHello.	23:4
Handshake Protocol	Server Certificate	The server sends a digital certificate to the client.	23:5
	Server Key Exchange	The server sends the Server Certificate to the client as a supplement to the Server Certificate.	23:6
	Certificate Request	The server requests a digital certificate from the client.	23:7
	Server Hello Done	Server sends a message to the client indicating the end of ServerHello.	23:8
	Client Certificate	The client sends a digital certificate to the server when required by the server.	23:9
	Client Key Exchange	The client sends pre-master secret to the server.	23:10
	Certificate Verify	Server authenticates the digital certificate of the client.	23:11
	Finished (encrypted handshake message)	Indicates that the authentication succeeds, usually after a ChangeCipherSpec message.	23:12
Application Data Protocol	Application data	Data transmission.	24:

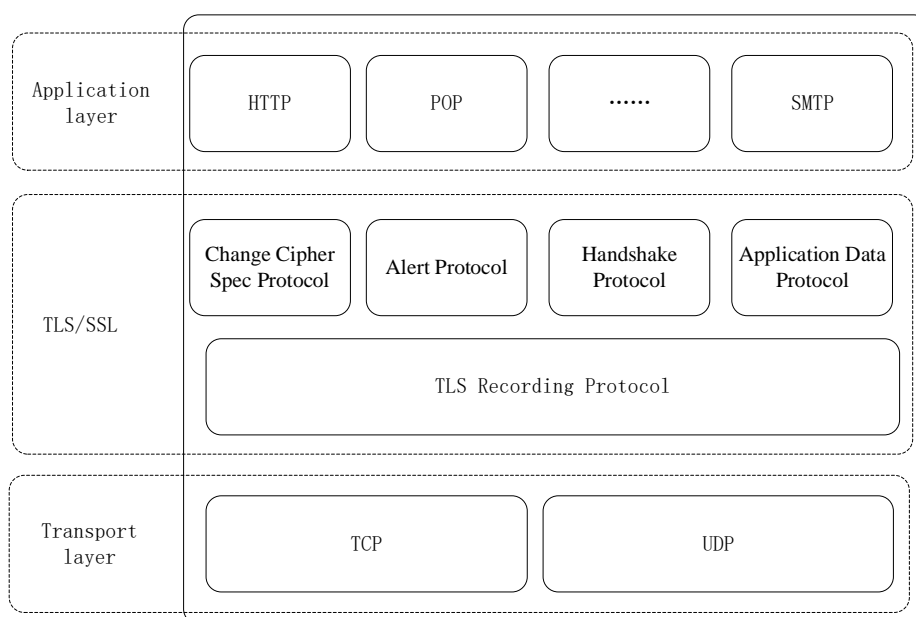


Figure 4. The architecture of protocols.

The protocols in Figure 4 are labeled for description, as shown in Table 2.

Table 2. Protocols’notation.

Network Layer	Protocol	Notation	
Application layer (30:)	HTTP	31:	
	POP	32:	
	IMAP	33:	
TLS/SSL (20:)	Change Cipher Spec Protocol	21:	
	Alert Protocol	22:	
	Handshake Protocol	23:	
	Application Data Protocol	24:	
Transport layer (10:)	TCP(11:)	(SYN)	11:01
		(ACK)	11:02
		(SYN, ACK)	11:03
		(FIN, ACK)	11:04
		(PSH, ACK)	11:05
		(RST, ACK)	11:06
	UDP	12:	

This section describes the communication process between the client and server by analyzing TLS protocol contents and analyzing communication messages. After marking each message, one can prepare for fingerprinting extraction.

3.2. Network Traffic Segment

Packets constitute the data in the network, and each packet has a fixed attribute. Figure 5 shows a segment of network traffic captured in the actual QQ communication process, which also contains some data packets from other applications.

In the actual process of obtaining traffic, the traffic packets sent and received by the terminal must be a mixture of multiple applications, which makes it possible to divide traffic based on IP address transition (IAT).

The network traffic obtained here contains a large number of encrypted packets, which can be directly obtained by processing only six limited attributes: time, source, destination, protocol, length, and info. Here, to define any packets the following is used: $p_i = [t, I_{src}, IP_{dst}, prt, l, Info]$. Based on the data packet attributes that can be directly obtained, this section analyzes the sequence of data packets to obtain encrypted traffic fingerprints.

Data packets generated when multiple applications installed on a terminal exchange data with the server are sent sequentially. Within a time interval $T = [t_{start}, t_{end}]$, define the data flow $T_f = (p_1, p_2, \dots, p_N)$ generated (N is the number of packets obtained). Any packet contains six attributes: $p_i = [t, IP_{src}, IP_{dst}, prt, l, Info]$.

The object to be dealt with is the data flow defined above. Given a data flow T_f of mixed data from multiple applications, the proposed method should accomplish two tasks:

- Process the data stream T_f and divide the packet sequence $seq_T = \{p_1, p_2, \dots, p_N\}$ into several small data stream segments according to the relevant attributes and message analysis:

$$S_{T_f} = \left\{ \langle p_1, p_2, \dots, p_1 \rangle, \langle p_{i+1}, p_{i+2}, \dots, p_j \rangle, \dots, \langle p_{j+1}, p_{j+2}, \dots, p_N \rangle \right\}$$

- Based on data stream fragments, the dynamic extraction of fine-grained fingerprint features is realized through the method. The specific method is described in Section 3.

The extracted application fingerprint is a message sequence containing encrypted messages: $seq_M = msg_1, msg_2, \dots, msg_M$.

No.	Time	Source	Destination	Protocol	Length	Info
325	0.928,778	202.108.3.242	192.168.155.2	TCP	66	110 → 55,996 [ACK] Seq = 35 Ack = 46 Win = 14,592 Len = 0 TSval = 1431,412,556 TSecr = 219,223,722
326	0.930,263	202.108.3.242	192.168.155.2	TCP	66	110 → 55,995 [ACK] Seq = 35 Ack = 44 Win = 14,592 Len = 0 TSval = 2632,709,893 TSecr = 219,223,722
327	0.930,725	182.254.106.125	192.168.155.2	TCP	66	80 → 56,008 [ACK] Seq = 1 Ack = 538 Win = 15,872 Len = 0 TSval = 185,319,318 TSecr = 219,223,783
328	0.935,148	182.254.50.144	192.168.155.2	TCP	135	8080 → 56,007 [PSH, ACK] Seq = 632 Ack = 5554 Win = 31,744 Len = 81
329	0.938,977	192.168.155.2	182.254.50.144	TCP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 713 Win = 262,048 Len = 0
330	0.947,171	182.254.50.144	192.168.155.2	TCP	293	8080 → 56,007 [PSH, ACK] Seq = 713 Ack = 5554 Win = 31,744 Len = 239
331	0.949,187	192.168.155.2	182.254.50.144	TCP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 952 Win = 261,888 Len = 0
332	0.952,219	182.254.106.125	192.168.155.2	TCP	1414	80→56,008 [ACK] Seq = 1 Ack = 538 Win = 15,872 Len = 1348 TSval = 185,319,323 TSecr = 219,223,783 [TCP segment of a reassembled PDU]
333	0.952,316	182.254.106.125	192.168.155.2	HTTP	106	HTTP/1.1 200 OK (text/plain)
334	0.959,318	192.168.155.2	182.254.106.125	TLSv1	66	56,008 → 80 [ACK] Seq = 538 Ack = 1389 Win = 130,688 Len = 0 TSval = 219,223,858 TSecr = 185,319,323
335	1.007,010	192.168.155.2	202.4.130.127	TLSv1	119	Application Data
336	1.012,887	202.4.130.127	192.168.155.2	TCP	231	Application Data
337	1.015,259	192.168.155.2	202.4.130.127	TLSv1	66	55,997 → 993 [ACK] Seq = 419 Ack = 1051 Win = 130,880 Len = 0 TSval = 219,223,917 TSecr = 3629,443,711
338	1.026,989	192.168.155.2	202.4.130.127	TLSv1	188	Application Data, Application Data
339	1.040,356	202.4.130.127	192.168.155.2	TCP	119	Application Data
340	1.043,178	192.168.155.2	202.4.130.127	TLSv1	66	55,997 → 993 [ACK] Seq = 541 Ack = 1104 Win = 131,008 Len = 0 TSval = 219,223,944 TSecr = 3629,443,736
341	1.043,263	192.168.155.2	202.4.130.127	TLSv1	156	Application Data, Application Data
342	1.056,409	202.4.130.127	192.168.155.2	TCP	231	Application Data
343	1.063,364	182.254.50.144	192.168.155.2	TCP	1494	8080→56,007 [ACK] Seq = 952 Ack = 5554 Win = 31,744 Len = 1440
344	1.063,829	182.254.50.144	192.168.155.2	TCP	829	8080→56,007 [PSH, ACK] Seq = 2392 Ack = 5554 Win = 31,744 Len = 775
345	1.068,176	192.168.155.2	202.4.130.127	TCP	66	55,997 → 993 [ACK] Seq = 631 Ack = 1269 Win = 130,880 Len = 0 TSval = 219,223,959 TSecr = 3629,443,753
346	1.068,26	192.168.155.2	182.254.50.144	POP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 3167 Wlen = 261,344 Len = 0
347	1.175,197	202.108.3.242	192.168.155.2	TCP	93	S: →OK 0 messages (0 octets)
348	1.178,540	192.168.155.2	202.108.3.242	POP	66	55,995→110 [ACK] Seq = 44 Ack = 62 Win = 132,032 Len = 0 TSval = 219,224,076 TSecr = 2632,710,132
349	1.184,621	202.108.3.242	192.168.155.2	DNS	97	s: →OK 1 messages (25,625 octets)
350	1.190,413	192.168.155.1	192.168.155.1	TCP	72	Standard query 0xc290 A 3ging.qq.com
351	1.191,133	192.168.155.2	202.108.3.242	DNS	66	55,996 → 110 [ACK] Seq = 46 Ack = 66 Win = 132,032 Len = 0 TSval = 219,224,085 TSecr = 1431,412,812
352	1.200,221	192.168.155.1	192.168.155.2	TCP	544	Standard query response 0xc290 A 3ging.qq.com CHNAME p31yuan2. tc . qq. com CNAME x2. tc . qq. com CNAME 56,009 → 80 [SYN] Seq = 0 Win = 65,535 Len = 0 MSS = 1460 WS = 32 TSval = 219,224,128 TSecr = 0 SACK_PERM = 1
353	1.231,191	192.168.155.2	103.18.209.13	TCP	78	56,009 → 80 [SYN, ACK] Seq = 0 Ack = 1 Win = 14,600 Len = 0 MSS = 1448 SACK_PERM = 1 W5 = 128
354	1.243,376	103.18.209.13	192.168.155.2	TCP	66	80 → 56,009 [SYN, ACK] Seq = 0 Ack = 1 Win = 14,600 Len = 0 MSS = 1448 SACK_PERM = 1 W5 = 128
355	1.246,084	192.168.155.2	103.18.209.13	TCP	54	56,009→80 [ACK] Seq = 1 Ack = 1 Win = 262,144 Len = 0
356	1.271,214	192.168.155.2	182.25450.144	TCP	1017	56,007 → 8080 [PSH, ACK] Seq = 5554 Ack = 3167 Win = 262,144 Len = 963
357	1.299,043	182.254.50.144	192.168.155.2	TCP	1494	8080 → 56,007 [ACK] Seq = 3167 Ack = 6517 Win = 34,560 Len = 1440
358	1.299,146	182.254.50.144	192.168.155.2	TCP	189	8080 → 56,007 [PSH, ACK] Seq = 4607 Ack = 6517 Win = 34,560 Len = 135 [TCP segment of a reassembled PDU]
359	1.302,202	192.168.155.2	182.254.50.144	TCP	54	56,007→8080 [ACK] Seq = 6517 Ack = 4742 Win = 260,544 Len = 0
360	1.319,215	182.254.50.144	192.168.155.2	TCP	189	8080 → 56,007 [PSH, ACK] Seq = 4742 Ack = 6517 Win = 34,560 Len = 135
361	1.321,632	192.168.155.2	182.254.50.144	HTTP	54	56,007 → 8080 [ACK] Seq = 6517 Ack = 4877 Win = 261,984 Len = 0
362	1.406,141	192.168.155.2	103.18.209.13	TCP	337	GET /qq_product_operations/nettest/index.html?r = 934,585,103 HTTP/1.1
363	1.415,448	103.18.209.13	192.168.155.2	TCP	54	80→56,009 [ACK] Seq = 1 Ack = 284 Win = 15,744 Len = 0
364	1.415,960	103.18.209.13	192.168.155.2	TCP	437	80→56,009 [PSH, ACK] Seq = 1 Ack = 284 Win = 15,744 Len = 383 [TCP segment of a reassembled PDU]
365	1.416,278	103.18.209.13	192.168.155.2	HTTP	63	HTTP/1.1 200 OK (text/html)
366	1.417,007	192.168.155.2	103.18.209.12	TCP	78	56,010 → 80 [SYN] Seq = 0 Win = 65,535 Len = 0 MSS = 1460 WS = 32 TSval = 219,224,305 TSecr = 0 SACK_PERM = 1
367	1.418,202	192.168.155.2	103.18.209.13	TCP	54	56,009→80 [ACK] Seq = 284 Ack = 384 Win = 261,760 Len = 0

Figure 5. Encrypted communication process.

This paper proposes a new method for the division of data flow T_f , which is no longer based on the simple time interval but on the six basic attributes of packets to find the “breakpoint” of network traffic, to achieve the purpose of accurate division of network traffic.

In the process of sending and receiving data packets, network devices complete an “action” in a time interval, similar to that in an operating system. The CPU is divided into several time slices, and each time slice processes one process. This “action” is equivalent to the application fingerprint that can be extracted. Therefore, in dividing data flows, the determination of “breakpoints” is based on this feature of network devices processing data. The “breakpoint” is determined according to the partitioning performance of each attribute of network traffic, that is, the optimal partitioning attribute. Among the six basic attributes shared by network packets, three attributes can be used as network traffic partitioning: time, source IP address, and destination IP address. In this paper, network traffic is considered as a whole to determine the “breakpoint”, and the partition performance of these three attributes in dividing network traffic is proved from the perspective of information entropy in information theory.

This paper introduces information entropy to calculate the information value contained in a sample. The basic idea is that if each element in a sample has a specific rule, it can be determined according to statistical methods, and its change content can be predicted in advance, which means that the information value of this sample is not high, and the information entropy contained in it is small. On the contrary, if all elements of the sample are disordered and the specific situation is unpredictable, the amount of information

contained in the sample will be large, and the information entropy will naturally be larger. Equation (1) is the definition of information entropy:

$$H(X) = - \sum_{x \in X} p(x) \log p(x) \tag{1}$$

The network traffic classification process is to divide the packets according to the properties of continuous discretization into information entropy smaller fragments, and each fragment contains less information entropy, which means that the element is regular and will repeat. This is the same as the nature of the application of a fingerprint; the information entropy is small and it will repeat traffic division and be representative. In addition, it can be used as an application fingerprint to identify traffic. Information gain is used as the measurement standard when selecting partition attributes. Information gain is the change that occurs before and after the dataset is partitioned, and (2) is the definition of information gain:

$$Info_G = H(X) - \sum_{i \in X} i \cdot H(X)_i \tag{2}$$

The classification process can be described in a tree structure, in which $H(X)$ represents the entropy of attributes of the parent node and $H(X)_i$ represents the entropy of attributes of several child nodes. The larger the information gain $Info_G$ is, the more information is contained before classification. The more impure the sample elements are, the smaller the information entropy after classification is, and the purer the sample elements of the child nodes will be. Each sample is more representative of the applied fingerprint, so the division based on IP address is better than the division based on time.

In network traffic, network packets are continuous-discrete samples and network traffic is T_f packet sequence $seq_T = \{p_1, p_2, \dots, p_i, \dots, p_N\}$, while $p_i = (\Delta t_i, \{IP_{src}, IP_{dst}\})$, Δt_i is the time difference between adjacent packets and $\{IP_{src}, IP_{dst}\}$ is the IP addresses at both ends of communication for unified analysis, regardless of the order. Therefore, according to the method of time interval division, the information gain of the obtained samples is:

$$Info_G(T) = H(\Delta T) - \sum_{i \in p_i} i \Delta H(\{IP_{src}, IP_{dst}\}) \tag{3}$$

$H(\Delta T)$ is the network traffic time interval of information entropy, as shown in (4):

$$H(\Delta T) = - \sum_{\Delta t_i \in T} p(\Delta t_i) \log p(\{\Delta t_i\}) \tag{4}$$

$H(\{IP_{src}, IP_{dst}\})$ is the information entropy in the i th partition, which can be calculated as follows:

$$H(\{IP_{src}, IP_{dst}\}) = - \sum_i p(MT_i) \log p(MT_i) \tag{5}$$

Another division method is based on IP address division $\{IP_{src}, IP_{dst}\}$. The calculation method is as follows:

$$Info_G(\{IP_{src}, IP_{dst}\}) = H(\{IP_{src}, IP_{dst}\}) - \sum_{i \in \Delta T} i \cdot H(\Delta T_i) = - \sum_i p(MT_i) \log p(MT_i) - \sum_{i \in p_i} i \cdot (- \sum_{\Delta t_i \in T} p(\Delta t_i) \log p(\Delta t_i)) \tag{6}$$

In the actual division process, only the data flow is divided into small fragments. The information entropy only proves that the method of division by IP address proposed in this paper, IAT, is more complete and accurate than that by time interval. Specific results of two different division methods for network data flow are calculated in the experimental part based on actual network communication data according to Formulas (3) and (6).

The following defines IAT: Given a data stream $seq_T = \{p_1, p_2, \dots, p_i, p_j, \dots, p_N\}$, any data packet p_i has an attribute time and IP address pair $(\Delta t_i, \{IP_{src}, IP_{dst}\})$. If any of the IP address pairs $\{IP_{src}, IP_{dst}\}$ of two adjacent packets p_i, p_j change, the data stream is

divided into two $\{p_1, p_2, \dots, p_i, p_j, p_{j+1}, \dots, p_j\}$ segments. The sequence exchange of IP addresses is not considered as the change of IP addresses.

A segment of network traffic obtained by IP address is classified into N -gram types based on message types. Here, it is first expressed in sequence, and each message is represented by a label defined above. For example, 42 packets in Figure 3 can be divided into 17 segments, as shown in Figure 6.

No.	Time	Source	Destination	Protocol	Length	Info
325	0.928,778	202.108.3.242	192.168.155.2	TCP	66	110 → 55,996 [ACK] Seq = 35 Ack = 46 Win = 14,592 Len = 0 TSval = 1431,412,556 TSecr = 219,223,722
326	0.930,263	202.108.3.242	192.168.155.2	TCP	66	110 → 55,995 [ACK] Seq = 35 Ack = 44 Win = 14,592 Len = 0 TSval = 2632,709,893 TSecr = 219,223,722
327	0.930,725	182.254.106.125	192.168.155.2	TCP	66	80 → 56,008 [ACK] Seq = 1 Ack = 538 Win = 15,872 Len = 0 TSval = 185,319,318 TSecr = 219,223,783
328	0.935,148	182.254.50.144	192.168.155.2	TCP	135	8080 → 56,007 [PSH, ACK] Seq = 632 Ack = 5554 Win = 31,744 Len = 81
329	0.938,977	192.168.155.2	182.254.50.144	TCP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 713 Win = 262,048 Len = 0
330	0.947,171	182.254.50.144	192.168.155.2	TCP	293	8080 → 56,007 [PSH, ACK] Seq = 713 Ack = 5554 Win = 31,744 Len = 239
331	0.949,187	192.168.155.2	182.254.50.144	TCP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 952 Win = 261,888 Len = 0
332	0.952,219	182.254.106.125	192.168.155.2	TCP	1414	80→56,008 [ACK] Seq = 1 Ack = 538 Win = 15,872 Len = 1348 TSval = 185,319,323 TSecr = 219,223,783 [TCP segment of a reassembled PDU]
333	0.952,316	182.254.106.125	192.168.155.2	HTTP	106	HTTP/1.1 200 OK (text/plain)
334	0.959,318	192.168.155.2	182.254.106.125	TLSv1	66	56,008 → 80 [ACK] Seq = 538 Ack = 1389 Win = 130,688 Len = 0 TSval = 219,223,858 TSecr = 185,319,323
335	1.007,010	192.168.155.2	202.4.130.127	TLSv1	119	Application Data
336	1.012,887	202.4.130.127	192.168.155.2	TCP	231	Application Data
337	1.015,259	192.168.155.2	202.4.130.127	TLSv1	66	55,997 → 993 [ACK] Seq = 419 Ack = 1051 Win = 130,880 Len = 0 TSval = 219,223,917 TSecr = 3629,443,711
338	1.026,989	192.168.155.2	202.4.130.127	TLSv1	188	Application Data, Application Data
339	1.040,356	202.4.130.127	192.168.155.2	TCP	119	Application Data
340	1.043,178	192.168.155.2	202.4.130.127	TLSv1	66	55,997 → 993 [ACK] Seq = 541 Ack = 1104 Win = 131,008 Len = 0 TSval = 219,223,944 TSecr = 3629,443,736
341	1.043,263	192.168.155.2	202.4.130.127	TLSv1	156	Application Data, Application Data
342	1.056,409	202.4.130.127	192.168.155.2	TCP	231	Application Data
343	1.063,364	182.254.50.144	192.168.155.2	TCP	1494	8080→56,007 [ACK] Seq = 952 Ack = 5554 Win = 31,744 Len = 1440
344	1.063,829	182.254.50.144	192.168.155.2	TCP	829	8080→56,007 [PSH, ACK] Seq = 2392 Ack = 5554 Win = 31,744 Len = 775
345	1.068,176	192.168.155.2	202.4.130.127	TCP	66	55,997 → 993 [ACK] Seq = 631 Ack = 1269 Win = 130,880 Len = 0 TSval = 219,223,959 TSecr = 3629,443,753
346	1.068,26	192.168.155.2	182.254.50.144	POP	54	56,007 → 8080 [ACK] Seq = 5554 Ack = 3167 Win = 261,344 Len = 0
347	1.175,197	202.108.3.242	192.168.155.2	TCP	93	S: *OK 0 messages (0 octets)
348	1.178,540	192.168.155.2	202.108.3.242	POP	66	55,995→110 [ACK] Seq = 44 Ack = 62 Win = 132,032 Len = 0 TSval = 219,224,076 TSecr = 2632,710,132
349	1.184,621	202.108.3.242	192.168.155.2	DNS	97	s: *OK 1 messages (25,625 octets)
350	1.190,413	192.168.155.2	192.168.155.1	TCP	72	Standard query 0xc290 A 3gimg.qq.com
351	1.191,133	192.168.155.2	202.108.3.242	DNS	66	55,996 → 110 [ACK] Seq = 46 Ack = 66 Win = 132,032 Len = 0 TSval = 219,224,085 TSecr = 1431,412,812
352	1.200,221	192.168.155.1	192.168.155.2	TCP	544	Standard query response 0xc290 A 3gimg.qq.com CHNAME p31yuan2. tc . qq. com CNAME p31yuan2. tc. qq. com CNAME x2. tc . qq. com CNAME
353	1.231,191	192.168.155.2	103.18.209.13	TCP	78	56,009 → 80 [SYN] Seq = 0 Win = 65,535 Len = 0 MSS = 1460 WS = 32 TSval = 219,224,128 TSecr = 0 SACK_PERM = 1
354	1.243,376	103.18.209.13	192.168.155.2	TCP	66	80 → 56,009 [SYN, ACK] Seq = 0 Ack = 1 Win = 14,600 Len = 0 MSS = 1448 SACK_PERM = 1 W5 = 128
355	1.246,084	192.168.155.2	103.18.209.13	TCP	54	56,009→80 [ACK] Seq = 1 Ack = 1 Win = 262,144 Len = 0
356	1.271,214	192.168.155.2	182.25450.144	TCP	1017	56,007 → 8080 [PSH, ACK] Seq = 5554 Ack = 3167 Win = 262,144 Len = 963
357	1.299,043	182.254.50.144	192.168.155.2	TCP	1494	8080 → 56,007 [ACK] Seq = 3167 Ack = 6517 Win = 34,560 Len = 1440
358	1.299,146	182.254.50.144	192.168.155.2	TCP	189	8080 → 56,007 [PSH, ACK] Seq = 4607 Ack = 6517 Win = 34,560 Len = 135 [TCP segment of a reassembled PDU]
359	1.302,202	192.168.155.2	182.254.50.144	TCP	54	56,007→8080 [ACK] Seq = 6517 Ack = 4742 Win = 260,544 Len = 0
360	1.319,215	182.254.50.144	192.168.155.2	TCP	189	8080 → 56,007 [PSH, ACK] Seq = 4742 Ack = 6517 Win = 34,560 Len = 135
361	1.321,632	192.168.155.2	182.254.50.144	HTTP	54	56,007 → 8080 [ACK] Seq = 6517 Ack = 4877 Win = 261,984 Len = 0
362	1.406,141	192.168.155.2	103.18.209.13	TCP	337	GET /qq_product_operations/nettest/index.html?r = 934,585,103 HTTP/1.1
363	1.415,448	103.18.209.13	192.168.155.2	TCP	54	80→56,009 [ACK] Seq = 1 Ack = 284 Win = 15,744 Len = 0
364	1.415,960	103.18.209.13	192.168.155.2	TCP	437	80→56,009 [PSH, ACK] Seq = 1 Ack = 284 Win = 15,744 Len = 383 [TCP segment of a reassembled PDU]
365	1.416,278	103.18.209.13	192.168.155.2	HTTP	63	HTTP/1.1 200 OK (text/html)
366	1.417,007	192.168.155.2	103.18.209.12	TCP	78	56,010 → 80 [SYN] Seq = 0 Win = 65,535 Len = 0 MSS = 1460 WS = 32 TSval = 219,224,305 TSecr = 0 SACK_PERM = 1
367	1.418,202	192.168.155.2	103.18.209.13	TCP	54	56,009→80 [ACK] Seq = 284 Ack = 384 Win = 261,760 Len = 0

Figure 6. IAT traffic segment.

Some of the fragments in Figures 2–5 contain only one data packet. Most of such fragments are caused by the out-of-order and retransmission of data sent by the terminal. In this way, the obtained fragments of network traffic are meaningless and are represented by labels in Tables 1 and 2, as shown in Table 3.

3.3. DNS Domain Text Clustering

The Domain Name System (DNS) is a database that maps IP addresses to domain names. The DNS server performs this function. Among the obtained network data packets, DNS messages are a few unencrypted messages with certain meanings. The domain names of DNS packets are extracted for text analysis, which can realize automatic network traffic analysis and largely replace some manual analysis. In addition, the corresponding URL can be quickly obtained in traffic classification and anomaly detection, which is convenient for further manual analysis and control in the massive network data flow. Part of the name is derived from the actual meaning of natural language, and part can be used as a combination of letters to represent a certain meaning. By mining domain name information,

one can extract effective text information from the domain name, which is of great help to the analysis of encrypted traffic.

Table 3. Network traffic segment.

Number	Network Traffic Segment
1	<11 : 02, 11 : 02>
2	<11 : 02>
3	<11 : 05, 11 : 02, 11 : 05, 11 : 02>
4	<11 : 02, 31 : 05, 11 : 02>
5	<24 :, 24 :, 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 :>
6	<11 : 02, 11 : 05>
7	<11 : 02>
8	<11 : 02>
9	< 32 :, 11 : 02, 32 :>
10	<->
11	<11 : 02>
12	<->
13	<11 : 01, 11 : 03, 11 : 02>
14	<11 : 05, 11 : 02, 11 : 05, 11 : 02, 11 : 05, 11 : 02>
15	<31 : 05, 11 : 02, 11 : 05, 31 : 05>
16	<11 : 01>
17	<11 : 02>

This section analyzes the characteristics of domain names of each application category and extracts representative domain names to provide domain name data for exception analysis and further manual exception analysis. Domain name analysis uses incremental clustering to process dynamic data streams, including RS extraction, and generation of related-word set, feature vector generation, and domain name clustering. Figure 7 shows the process.

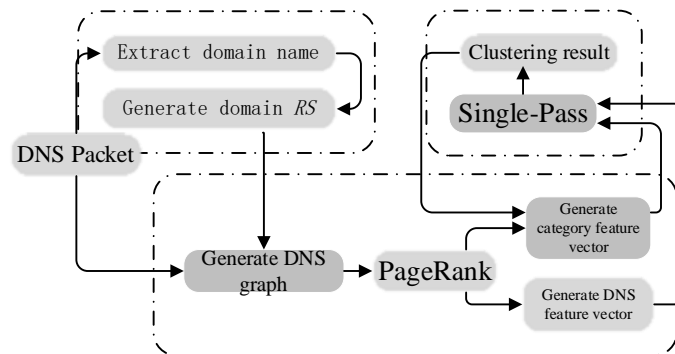


Figure 7. DNS text clustering.

Given a domain name as shown in “ $str_1.str_2.\dots.str_n$ ”, according to “.” to extract string of “ $str_1.str_2.\dots.str_n$ ” to generate string set, defined as an item in domain name correlation set RS .

Network data flows usually generate multiple DNS messages, so RS consists of multiple components, $RS = \{RS^1, RS^2, \dots, RS^i, \dots\}$, and because the new DNS message is generated continuously, RS will continue to expand. For one type of other DNS-message-generated domain correlation set, $RS = \{RS^1, RS^2, \dots, RS^i, \dots\}$, based on the RS built DNS, DNS is graphed with the vertex and edge of the relationship between domain name and domain names. In the DNS graph, vertices represent strings in domain names, and edges represent relationships between domain names. Edges are added according to the domain name correlation set RS . In a domain name correlation set, edges can be added, and the direction is from the high-level domain name to the lower-level domain name, thus forming the directed DNS graph.

Different from natural language, the domain name can be any combination of the characters, so as the network traffic constantly produces, there will be a new domain name constantly generated, which means that the new node will continue to produce to solve this problem, and to a certain extent reduce the computational complexity, as well as needs to obtain a large number of DNS messages and complete the extraction of node information. After the DNS graph is established, the importance of nodes is used as the feature vector of DNS messages.

The PageRank algorithm was first used in the Google search engine. In order to accurately rank web pages, the number and quality of links in web pages are used to measure the importance of web pages. The directed graph $G = (V, E)$ constructed by DNS messages is similar to the web-page ranking in search engines, in order to find important node information. Assume that n is the number of nodes, V and E represent vertices and edges, and the PageRank value $G(i)$ of vertex i is shown in (7):

$$G(i) = \sum_{(i,j) \in E} \frac{G(j)}{K_j} \tag{7}$$

Calculate the PageRank value of each node, and use G to represent the PageRank value of all nodes, as shown in (8):

$$G = (G(1), G(2), \dots, G(n))^T \tag{8}$$

Assume that A is the adjacency matrix of graph G , as shown in (9):

$$A_{ij} = \begin{cases} \frac{1}{K_i}, & \text{if } (i,j) \in E \\ 0, & \text{if } (i,j) \notin E \end{cases} \tag{9}$$

Then, (7) can be converted into (10):

$$G = A^T G \tag{10}$$

In reality, the process of users surfing the Internet is relatively complex. The PageRank algorithm assumes that surfing the Internet is a random process. In order to make A meet this condition, Equation (10) is improved, as shown in (11):

$$G = ((1 - f) \frac{E}{n} + f A^T) G \tag{11}$$

E is the full 1 matrix, f is the attenuation factor, and (11) can be converted into:

$$G = (1 - f)e + f A^T G \tag{12}$$

Therefore, (7) can be expressed as follows:

$$G(i) = (1 - f) + f \sum_{(i,j) \in E} \frac{G(j)}{K_j} \tag{13}$$

The calculation process of PageRank is a cyclic process. Given an initial value, the PageRank value of the node is calculated. When the change is less than the threshold value, the calculation is stopped. Calculating the PageRank value of the domain name can extract important domain names, the appearance of these important domain names often has certain meaning, and new domain names through manual analysis can have important roles in helping in anomaly detection.

The generation of the network DNS message is continuous. In order to process the newly generated messages in real time, the single-pass incremental clustering algorithm is adopted. Firstly, the eigenvalues of vectors of each class are calculated as the cluster

centers, and the similarity is used as the judgment criterion whenever a new message is generated, as shown in (14):

$$SI(M, n) = \frac{|M \cap n|}{M} \tag{14}$$

M is the clustering center vector, and n is the feature of DNS messages. The classification of DNS messages is determined through similarity calculation. After a certain period, a new DNS is generated, and the clustering center is recalculated according to the clustering result.

4. Fine-Grained High-Utility Dynamic Fingerprinting Extraction

According to the preprocessing of the network traffic in Section 3, several segments of the network traffic are obtained. The N -gram model is used to represent the results after the partition, and the label is used to represent the results, as shown in Table 3. In this way, the partition results can retain the complete application fingerprint to the maximum extent. Network behavior is a constantly changing data flow. To obtain the changes in new network behavior in time, it is necessary to obtain application fingerprints dynamically online, which requires that network traffic can only be processed once.

In Section 4, the problem to be solved is that application fingerprints with typical features can be dynamically obtained. Based on a frequent set mining algorithm, a fine-grained efficient dynamic application fingerprint extraction method Huf is proposed. The Huf algorithm focuses on two parts: one is to calculate the utility of application fingerprints, and the other is to build an N -gram tree structure to mine application fingerprints with high-utility values according to the tree structure.

4.1. The Utility of Fingerprinting

The process of obtaining the most typical key exchange in the communication process can more accurately describe the communication process. The TCP Triple Handshake Protocol accounts for more than 80% of the total number of packets. It is difficult to analyze except for differences in length. The number of TLS-related messages is about 10%. Every time an encrypted communication process is established, the communication between the client and the server requires a complete key and certificate exchange process, which has a great impact on the analysis of communication behavior. That is, the smaller number of message types have strong behavior analysis capabilities. Therefore, the differentiation ability of such messages to data flows is defined as utility, which can be understood as the value of the message itself. Similar to natural language processing, there are some frequently used words in natural languages, such as “de”, “le”, and “me”, which are not helpful for the next algorithm.

Here, combined with the evaluation of the importance of vocabulary in natural language processing algorithms, the Term Frequency-Inverse Document Frequency (tf-idf) algorithm defines the utility.

In the sequence representation $s = \langle p_1, p_m, \dots, p_n \rangle$ of an N -gram model, the utility of a term p_m is defined as:

$$u(p_m) = \varphi(n_{p_m}, l) \times \phi(n_d, n_d^{p_m}) \tag{15}$$

The number of messages p_m in sequence s is n_{p_m} , and the length of s is l . The proportion of p_m in sequence s is $\varphi(n_{p_m}, l)$, which is defined as follows:

$$\varphi(n_{p_m}, n) = \frac{n_{p_m}}{n} \tag{16}$$

The total number of fingerprints is n_d , $n_d^{p_m}$ is the fingerprint containing the message p_m , and $\phi(n_d, n_d^{p_m})$ is defined as the following:

$$\phi(n_d, n_d^{p_m}) = \ln \frac{n_d}{n_d^{p_m} + 1} \tag{17}$$

A large number of data flow fragments in Table 3 are useless, for example, a data flow fragment with only one packet. Table 3 shows only a small part of the data flow. To illustrate the utility’s calculation, assume that the Table 3 below lists all the fragments of the data flow.

Table 4 describes the utility calculation process. When network traffic is collected for a long enough time, a large-scale utility using a fingerprint is calculated by a statistical method, which is similar to natural language. A large enough corpus ensures the accuracy of the tf-idf algorithm. Data stream fragments are directly associated with transactional databases in the frequent item set mining.

Table 4. Transaction database.

TID	Transaction Database (Item, Quantity)
1	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02 \rangle$
2	$\langle 23 : 3, 23 : 5, 23 : 8, 11 : 02, 23 : 10, 11 : 02, 21 :, 24 : \rangle$
3	$\langle 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 : \rangle$
4	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02, 23 : 3, 23 : 5, 23 : 8 \rangle$
5	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 23 : 3, 23 : 5, 23 : 8, 11 : 02, 23 : 10, 21 :, 23 : 12 \rangle$
6	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02, 23 : 10, 21 :, 23 : 12 \rangle$
7	$\langle 11 : 02, 24 :, 24 :, 11 : 02, 23 : 12 \rangle$
8	$\langle 21 :, 11 : 02, 24 :, 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 : \rangle$

Calculate the utility of message 23:2 in Table 4 for a fingerprint TID = 4:

$$u(23 : 2) = \varphi(n_{23:2}, n) \times \phi(n_d, n_d^{23:2}) = \frac{1}{8} \times \ln \frac{8}{5} = 0.575$$

The average utility of sequence $s = \langle p_1, p_m, \dots, p_n \rangle$ calculates the average utility of the whole fingerprint according to the utility of the message, which is defined as follows:

$$au(s) = \frac{\sum_{X \in \{l, m, \dots, n\}} u(P_X)}{l(p_X)} \tag{18}$$

The utility still calculates the entire fingerprint with the application fingerprint of TID = 4:

$$\begin{aligned} au(s_{TID=4}) &= \frac{u(11:01)+u(11:03)+u(11:02)+u(23:2)+u(11:02)+u(23:3)+u(23:5)+u(23:8)}{l(s_{TID=4})} \\ &= \frac{0+0+0+0.575+0.087+0.087+0.087}{8} = 0.105 \end{aligned}$$

Tables 5 and 6 show the calculation results of each fingerprint.

Table 5. The average-utility of transaction database.

TID	Transaction Database (Item, Quantity)	Average Utility
1	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02 \rangle$	0.051
2	$\langle 23 : 3, 23 : 5, 23 : 8, 11 : 02, 23 : 10, 11 : 02, 21 :, 24 : \rangle$	1.259
3	$\langle 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 : \rangle$	0.616
4	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02, 23 : 3, 23 : 5, 23 : 8 \rangle$	0.120
5	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 23 : 3, 23 : 5, 23 : 8, 11 : 02, 23 : 10, 21 :, 23 : 12 \rangle$	0.189
6	$\langle 11 : 01, 11 : 03, 11 : 02, 23 : 2, 11 : 02, 23 : 10, 21 :, 23 : 12 \rangle$	0.124
7	$\langle 11 : 02, 24 :, 24 :, 11 : 02, 23 : 12 \rangle$	0.386
8	$\langle 21 :, 11 : 02, 24 :, 11 : 02, 24 :, 24 :, 11 : 02, 24 :, 24 : \rangle$	0.450

Table 6. The utility of items.

Item	11:01	11:02	11:03	21:	23:2	23:3	23:5	23:8	23:10	23:12	24:
Utility	0	0	0	0.212	0.254	0.236	0.236	0.236	0.236	0.288	0.821

4.2. Constructing N-gram Huf-Tree of Huf Algorithm

Applications on a terminal device need to send data, so seemingly unrelated independent data packets are connected. This paper uses the method based on association rules to discover the relationship between independent packets. The number of data packets is huge and cannot be completed manually. Therefore, the mining method based on frequent sets proposed in this paper realizes the automatic acquisition of data packet sequences, that is, the behavior pattern of communication.

Apriori is the first classical frequent item set mining algorithm in the world, which laid the foundation for many classical algorithms later. The main process of the Apriori-like algorithm is to find all candidate sets, and then remove the candidate sets that do not meet the requirements by pruning [23]. This algorithm needs to scan the database many times to generate a large number of candidate sets, which has a large space and time complexity and has a great impact on the efficiency of the algorithm. To improve the process, scan the database only once, and reduce the space and time complexity, many kinds of studies have proposed improvement methods, which are generally summarized as after scanning the data objects once, creating an “intermediate storage” to temporarily store the selected candidate set. All subsequent analysis is directed to this “intermediate store”. The difference is that these documents use different structures to represent this “intermediate storage”, such as LQS-Tree [24], EFP-Tree [25], MED-Tree [26], WMFP-SW-Tree [27], IHAUI-Tree [28], MAS-Tree [29], and HUWAS-Tree [30]. These are all variants of FP-Tree in the FP-Growth algorithm, an algorithm adapted from Apriori. The dynamic mining algorithm of the frequent episode generally uses the tree structure to replace several scans of sample data.

The Apriori algorithm contains two thresholds, which are important indicators of the algorithm and are used to measure the selection of frequent item sets. One is minimum support, which represents the minimum probability of the occurrence of an item in a transaction database, and the other is minimum confidence, which defines the probability of the occurrence of another item in a set of items when one item is present. Association rules use these two indicators to determine the relationship between item sets, which can be understood as a certain relationship between two items when they appear together with a large probability.

In general, the process for defining frequent item set mining is as follows.

The transaction database $D = (I_1, I_2, \dots, I_n)$ represents a set of several items, and each item can be represented by TID. The support degree of frequent item set $FI = (X_1, X_2, \dots, X_m)$ in D is represented by the proportion of frequent item set X_1, X_2, \dots, X_m in $D = (I_1, I_2, \dots, I_n)$, that is, $P_{sup} = (X_1, X_2, \dots, X_m)$. The confidence is used in the next prune operation and represents the probability of the term X , i.e., P , when I contains Z . The Apriori algorithm is used to complete the mining of frequent item sets through continuous connection and pruning. The process of the Apriori algorithm is shown as follows:

1. Scan transaction database $D = (I_1, I_2, \dots, I_n)$, and a 1-item set generates item $FIS^1 = (FI_1^1, FI_2^1, \dots, FI_n^1)$ in the transaction database that represents compliance support.
2. Start the connection from a 2-item set. The new item set is connected to the existing item set:
 - (a) Suppose the $k-1$ -item set $FIS^{k-1} = (FI_1^{k-1}, FI_2^{k-1}, \dots, FI_l^{k-1})$ is connected to generate the k -item set $FIS^k = (FI_1^k, FI_2^k, \dots, FI_l^k)$.
 - (b) The confidence degree of the candidate k -item set $FIS^k = (FI_1^k, FI_2^k, \dots, FI_l^k)$ is calculated and pruned to remove the item set that does not meet the confidence degree.
3. Repeat the above steps to generate a new item set until there is only one item in the item set, then stop the loop.

One of the most important features of network data flow is the continuous generation of data packets, which is an incremental process. Many existing kinds of literature carry out

feature extraction on static datasets after obtaining datasets, and these methods obviously cannot realize the online dynamic processing and analysis of data packets. Dynamic analysis can extract new features in time and monitor network data in real-time, so the method proposed in this paper can extract features dynamically. Above, the communication process between the client and server is extracted by establishing a sequential transaction database of data flow. In this section, the method of processing dynamic data flow by constructing the Huf-Tree is mainly proposed. A threshold ϵ is first set to determine whether to retain a candidate, as defined below.

The minimum threshold of average utility determines whether to join the Huf-Tree, which is expressed in proportion:

$$mau = \frac{\epsilon \times SDU}{TID} \tag{19}$$

Assuming that the threshold " $\epsilon = 50\%$ " is selected, Table 6 is used as an example:

$$mau = \frac{\epsilon \times SDU}{TID} = \frac{1}{8} \times 50\% \times (0 + 0 + 0 + 0.212 + 0.254 + 0.236 + 0.236 + 0.236 + 0.288 + 0.821) = 0.157$$

The target data to be processed in this paper are network data flow, which can be regarded as an incremental process $InP = (SDB_1, SDB_2, \dots, SDB_n, \dots)$. The process of constructing the Huf-Tree is shown in Figure 8.

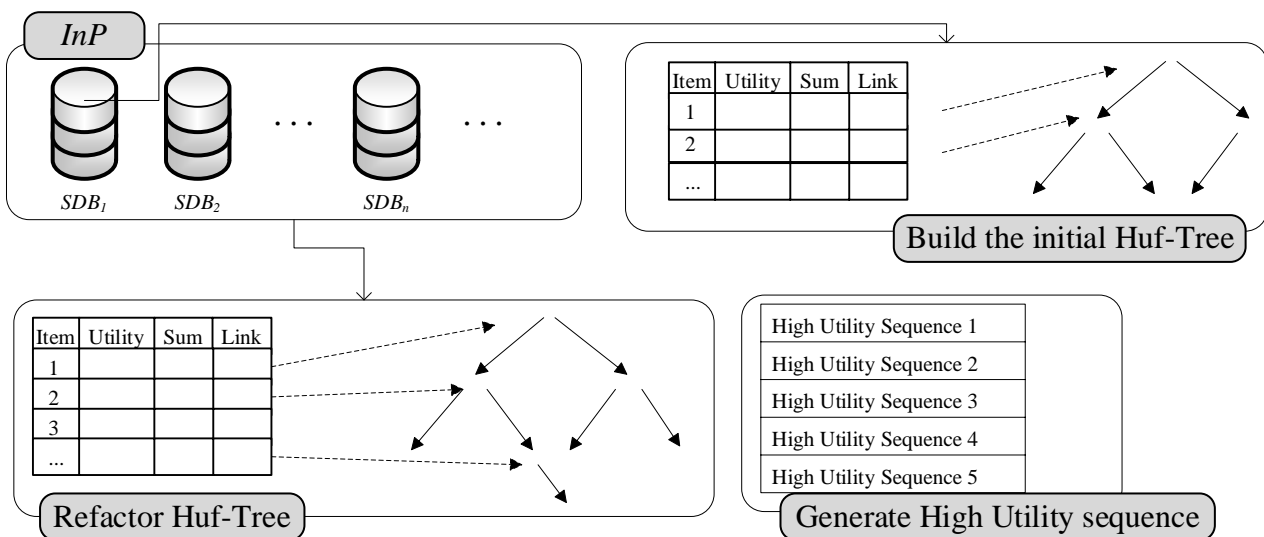


Figure 8. The construction of Huf-Tree.

The process shown in Figure 8 is the process of generating an application fingerprint high-utility sequence, which can be divided into four phases:

- The data are preprocessed and the incremental representation $InP = (SDB_1, SDB_2, \dots, SDB_n, \dots)$ of data flow is generated;
- The initial Huf-Tree is built, which takes SDB_1 as the initial database;
- When a new SDB_n is generated, each item is gradually added to update the Huf-Tree;
- New application fingerprints are generated based on the threshold at intervals.

The utility list is used to store the data related to the Huf-Tree. In essence, it is an array used to retrieve fingerprint $s = \langle p_1, p_2, \dots, p_n \rangle$.

The construction process of Huf-Tree is briefly introduced according to Table 7.

Table 7. Utility list.

Item	Utility	Sum	Link
11:01	0	1	-
11:02	0	7	-
11:03	0	1	-
11:03	0.212	1	-
23:2	0.254	1	-
23:3	0.236	1	-
23:5	0.236	1	-
23:8	0.236	1	-
23:10	0.288	1	-
23:12	0.236	-	-
24:	0.821	-	-

1. According to the data preprocessing part generated using a fingerprint, and gradually adding each path, the first path to add in the process is shown in Figure 9:

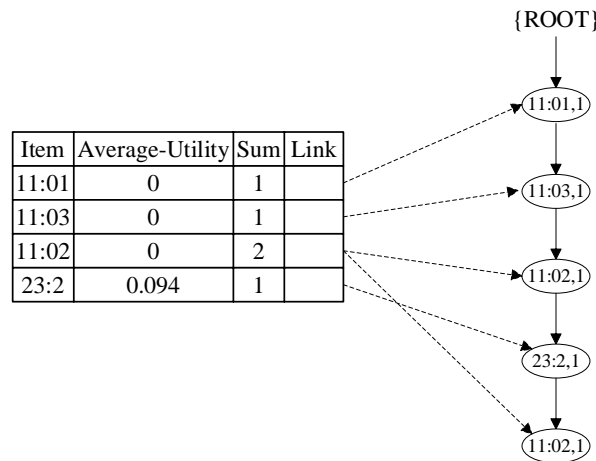


Figure 9. Huf-Tree.

2. Add each application fingerprint step by step. If there are duplicate fingerprints, some paths need to be merged, as shown in Figure 10.

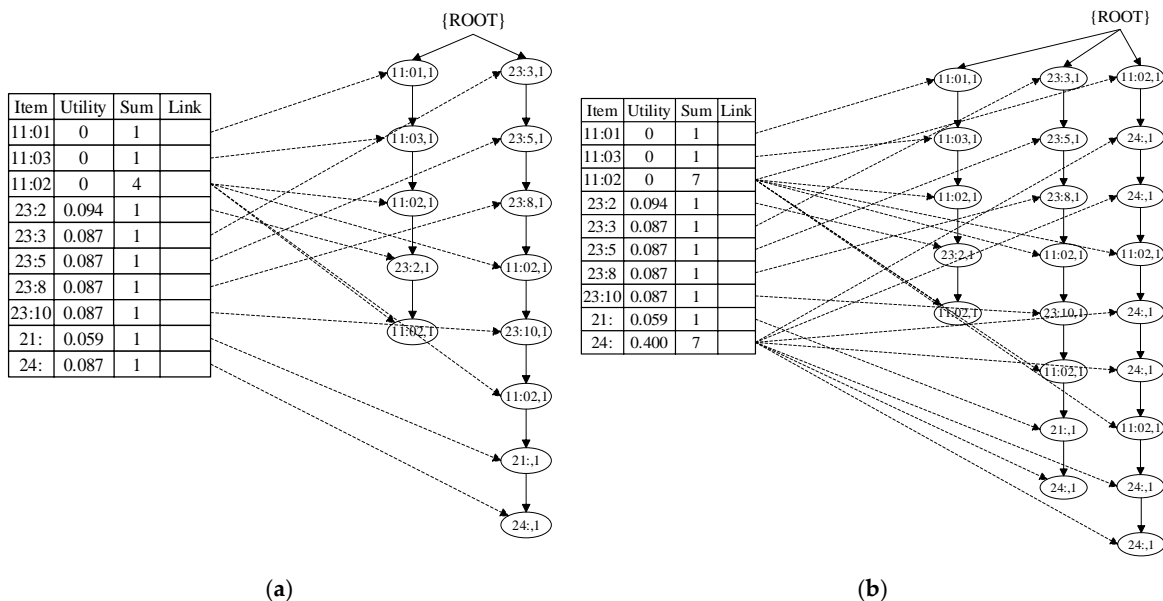


Figure 10. Cont.

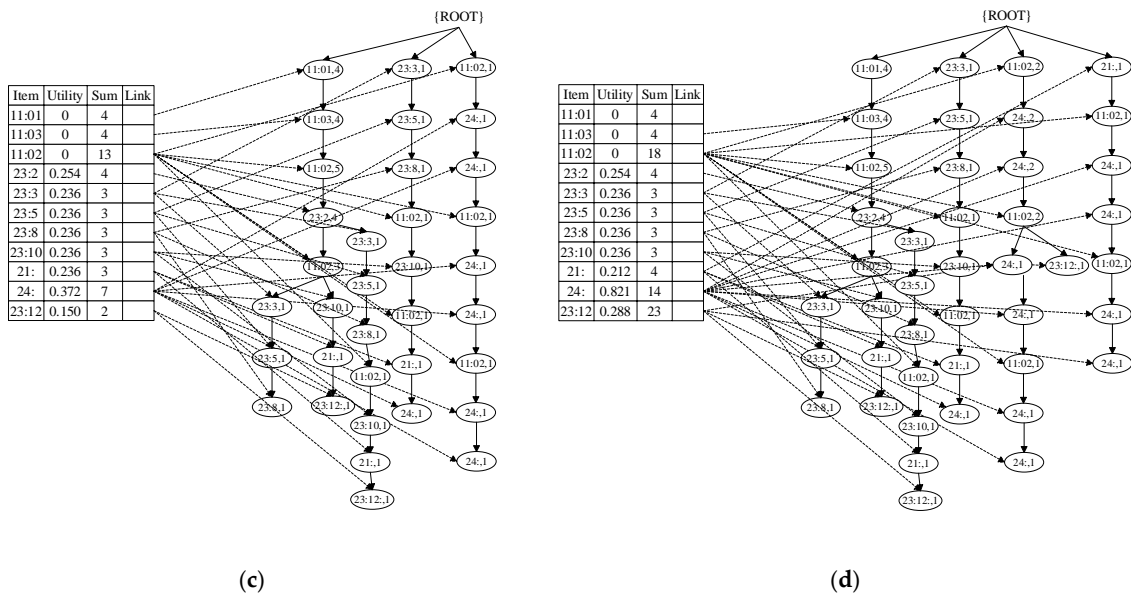


Figure 10. Refactor Huf-Tree. (a) Insert $S_{TID}=2$. (b) Insert $S_{TID}=3$. (c) Insert $S_{TID}=4,5,6$. (d) Insert $S_{TID}=7,8$.

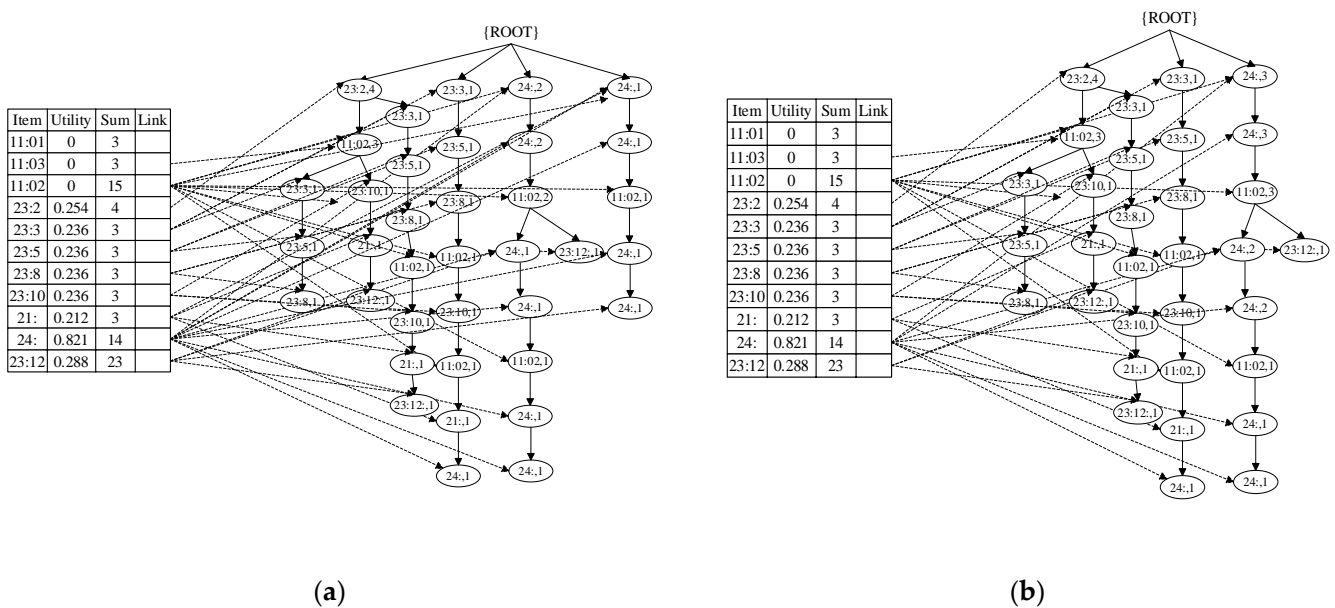
- Every once in a while, according to the preset threshold, the application fingerprint that meets the condition is extracted and the messages need to be sorted, as shown in Table 8:

Table 8. The ordered utility list.

Item	Utility	Sum
24:	0.821	14
23:2	0.254	4
23:3	0.236	3
23:5	0.236	3
23:8	0.236	3
23:10	0.236	3
23:12	0.288	23
21:	0.212	4
11:01	0	4
11:02	0	18
11:03	0	4

Based on the transaction database information shown in Table 8, find the paths where 24: is located: <23:3, 23:5, 23:8, 11:02, 23:10, 11:02, 21:, 24:>; <11:02, 24:, 24:, 11:02, 24:, 24:, 11:02, 24:, 24:>; and <21:, 11:02, 24:, 11:02, 24:, 24:, 11:02, 24:, 24:>. Then, calculate the average utility for each path, 0.249, 0.154, and 0.479, and keep <23:3, 23:5, 23:8, 11:02, 21:, 24:> and <21:, 11:02, 23:, 11:02, 24:, 24:, 11:02, 24:, 24:>.

It can be seen from Figure 10d that the structure of the Huf-Tree is very complex, which is only a Huf-Tree of a small section of the data stream. When new data flows are continuously processed, the generated Huf-Tree will be very large, so the structure of the Huf-Tree needs to be merged. Since each node of the Huf-Tree is based on the N -gram model, the order of nodes cannot be exchanged. Therefore, in the prune stage, this paper adopts two steps to prune: removing low-utility nodes and merging paths. The process is shown in Figure 11.



(a) Delete nodes. (b) Path merge.

Figure 11. Huf-Tree prune. (a) Delete nodes. (b) Path merge.

In order to ensure the integrity of the application fingerprint, the nodes in the middle of the application fingerprint cannot be deleted, but can only be deleted successively from the root node. Figure 11a is the result after deleting node <11:01, 11:02, 11:03, 21:>. Figure 11a merged the right-most path from the root node to the adjacent path. The same path of nodes is required to merge, with different nodes as children.

4.3. Dynamic Fingerprinting Extraction Algorithm Based on Huf-Tree

This section mainly proposes a dynamic application fingerprinting extraction algorithm based on Huf-Tree. The Huf algorithm mainly consists of three parts: data preprocessing, building Huf-Tree, and mining high-utility sequence, as shown in Figure 12:

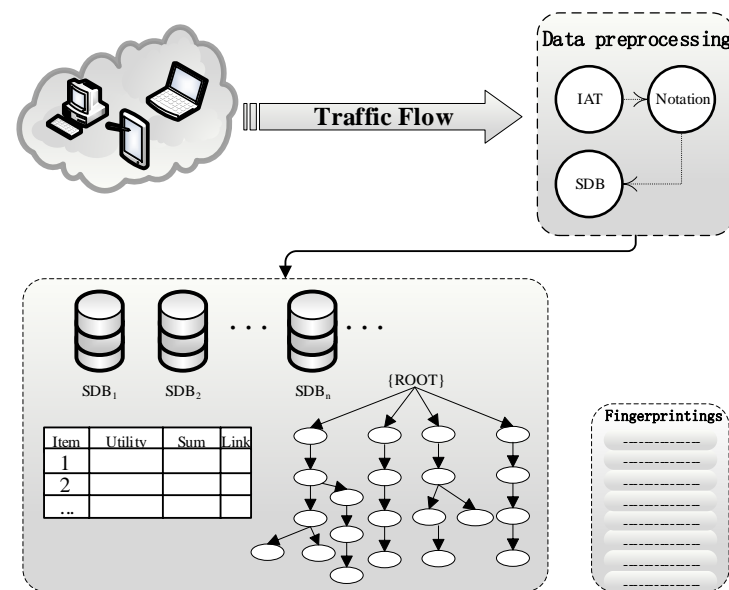


Figure 12. The process of Huf.

The specific algorithm is described in Algorithm 1, Algorithm 2 and Algorithm 3:

Algorithm 1: Data PreprocessingInput: network packet T_f ;Output: Application fingerprint library T_{tid} ;

- (1) IAT generates an application fingerprint: $T_f \rightarrow T_{IAT}$
- (2) Generate a fingerprint library based on the application fingerprint: $T_{IAT} \rightarrow T_{tid}$
- (3) Compute the utility of the messages in T_{tid} : $u(p_m), \varphi(n_{p_m}, l), \phi(n_d, n_d^{p_m})$;
- (4) Calculate the average utility of T_{tid} : $au(s)$.

Algorithm 2: HufInput: $SDB_1, SDB_2, \dots, SDB_n, \dots, \varepsilon$;Output: Huf-Tree T_{huf} ;

- (1) $SDB_1 = \langle p_1, p_m, \dots, p_n \rangle$, Generate the utility list: L_T ;
- (2) Create ROOT nodes and insert nodes in sequence: $ROOT \rightarrow T_{Huf}$;
- (3) Insert each fingerprint P : $p_1, p_m, \dots, p_n \rightarrow T_{Huf}$;
- (4) If the new fingerprint coincides with the existing path, it is merged with the existing path L_T ;
- (5) Delete nodes whose value is smaller than the threshold and connect their child nodes to ROOT;
- (6) The duplicate nodes of the ROOT node are merged into path P, and the different nodes become child nodes;

Repeat (1)~(6), insert SDB_2, \dots, SDB_n .**Algorithm 3:** Mining High UtilityInput: Huf-Tree $T_{huf}, L_T, \varepsilon$;Output: Fingerprint sequence F_p ;

- (1) Compute Utility of L_T , sort results to generate L'_T ;
- (2) Keep nodes greater than the threshold: $au(s) > \varepsilon$
- (3) Generate an application fingerprint: $F_p = (fp_1, fp_2, \dots, fp_n)$.

5. Experimental Analysis

This section is mainly based on the problem analysis, data preprocessing, and fingerprint extraction in the previous Sections 2–4. The following contents are divided into four parts. The first part is the non-invasive model, which only analyzes the situation of user equipment by grabbing network data without requiring the installation of specific software. The second part is the IAT traffic segment experiment, which performs information gain analysis of data flow division. The third part is the fingerprinting integrity experiment, mainly to analyze whether the acquired fingerprints are complete and accurate, and the fourth part is the high-utility fingerprinting extraction experiment.

5.1. Non-Invasive Model

Limited by hardware, mobile terminals represented by smartphones seldom install corresponding security software, and users sometimes doubt the monitoring software itself. Moreover, with the popularity of mobile payment, mobile terminals have greater risks. In order to complete the security monitoring of terminal devices without affecting the use of users and without requiring the installation of specific software, this paper proposes a non-invasive model that analyzes the situation of user devices only by capturing network data, as shown in Figure 13.

In the model shown in Figure 13, the monitor is connected to the Internet and the mobile device. The detector sends WiFi signals to enable the mobile device to communicate with the server. The monitor has a packet capture function and can obtain the communication data between the mobile terminal and the server. In practical application, the monitor

is generally an industrial computer integrated with wireless WiFi, wired network card, packet capture, and analysis functions. However, in the experimental analysis stage of the laboratory, computers and wireless network cards with promiscuous mode are used first, and packet capture is realized by Wireshark, which is also the packet capture tool used in the most relevant literature. The mobile device uses two Android mobile phones, which form the same structure as Figure 13. Wireshark is used for data processing, and the specific calculation is implemented by MATLAB and C programming.

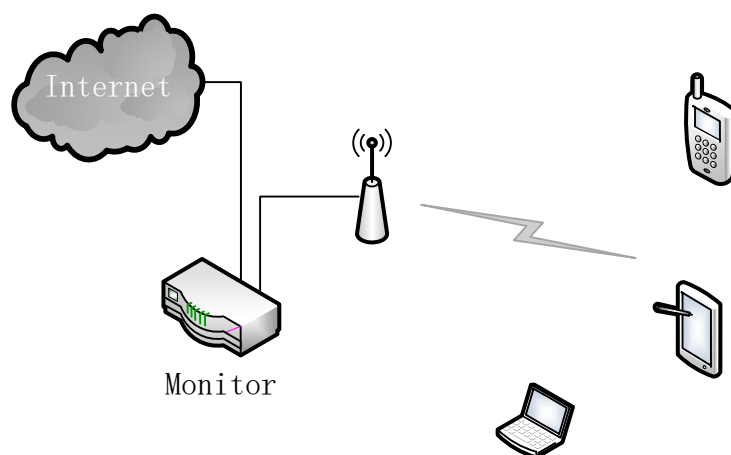


Figure 13. Non-invasive model.

The datasets adopted in the experiment are divided into two types. The first type is the public dataset, which comes from Netresec, a Swedish network security company. It published a lot of datasets, so only one dataset (WRCCDC-PCAPS from the Western Regional Collegiate Cyber Defense Competition) is selected as part of the public dataset in this experiment.

The second type of dataset is the network data captured by Wireshark in the actual environment. The experimental data captured are two smartphones equipped with the Android system, on which common software is installed. According to the classification of each application market, this paper installs the four most common categories of software on smartphones, including video, news, communication, and life, such as QQ, WeChat, email clients, the news client, etc. This paper puts forward the application for communication sessions through the frequent item set mining method of fingerprints. In the past there was no relevant research on this topic, so there was no ability for comparison with previous methods. This section outlines an experiment from the extracted fingerprint to analyze the integrity and accuracy of the analysis. The specific datasets are shown in Table 9.

Table 9. Dataset.

Name	Property or Category	Time (min)	Packet Number
Dataset1	Background traffic	30	1755
Dataset2	Video	5	115,106
Dataset3	News	5	139,193
Dataset4	Communication	5	73,912
Dataset5	Daylife	5	76,269
Dataset6	Mixed traffic	5	68,537
Dataset7	Mixed traffic	10	102,846
Dataset8	Mixed traffic	15	182,292
Dataset9	Public data	20	82,354
Dataset10	Public data	40	28,884

Dataset1 refers to the case that the smartphone obtains data packets when the user does not run any installation program. The smartphone sends a few data packets when it is

silent, mainly the push of some messages and the data sent by the operating system itself. The purpose of collecting such data is to observe the data used by non-users and measure the impact on the data used by users. There are only 1755 data points in 30 min, which is very few compared with the mixed data collected in 15 min, so this part of the data can be ignored. Dataset2, Dataset3, Dataset4, and Dataset5 are traffic collection situations that only run software of related categories. Dataset6, Dataset7, and Dataset8 are mixed traffic that run all applications to collect traffic. The difference is that the time is 5 min, 10 min, and 15 min. Dataset9 and Dataset10 are public datasets for online attack and defense competitions. The experiment consists of three parts: IAT traffic partition, application fingerprint extraction integrity analysis, and efficient application fingerprint extraction results analysis.

5.2. IAT Traffic Segment Experiment

The first step is traffic partitioning. According to the IAT address partitioning method proposed above, the following table shows the number of application fingerprints after partitioning. As can be seen from Table 10, the average length of a fingerprint has a relation with the number of packets and application category, such as that the video class fingerprint is long, because the video class sends traffic that is bigger, and users in general will finish loading the video they are watching. When a large amount of data are collected and more application analogies are used, the advantages of IAT address division will gradually expand, and the average fingerprint length will rapidly decrease. With a shorter fingerprint length, the extracted features will be more typical, and the traffic classification effect will be better.

Table 10. Dataset.

Name	Time (min)	Packet Number	Application Fingerprint Number	Average Fingerprint Length
Dataset1	30	1755	441	3.97
Dataset2	5	115,106	6830	16.85
Dataset3	5	139,193	9104	15.28
Dataset4	5	73,912	7643	9.67
Dataset5	5	76,269	6986	10.91
Dataset6	5	68,537	7366	9.30
Dataset7	10	102,846	13,857	7.42
Dataset8	15	182,292	149,683	1.21
Dataset9	20	82,354	7395	11.13
Dataset10	40	28,884	3735	7.73

As can be seen from Table 10, the information gain of data stream division by IAT is higher than that by time division. The reason is that the data stream divided by time has great contingency, the slice repeatability is small, the information entropy is still large, and the nodes are not pure. Therefore, the repeatability of the application fingerprint is very high, and only dozens of application fingerprints can be extracted from the possibly large data stream. Moreover, when the data volume is large enough, the more applications are used, the more frequent the switching and switching between addresses, the shorter the fingerprint obtained, the greater the possibility of repetition, the more pure the node information is, and the greater the information gain will be. This indicates that the application fingerprint classification effect is better.

First, calculate the information entropy of the entire network traffic, as shown in Formula (20).

$$H(M) = - \sum_{x \in M} p(x) \log p(x) = - \sum_{x \in M} \frac{N_x}{N} \log \frac{N_x}{N} \tag{20}$$

where M is the message type, including TCP, HTTP, POP, and TLS. Then, the information entropy of the whole data stream can be obtained by calculating the probability of different

types of messages under different protocols. First, the data packet is divided according to the IAT method, and then the information entropy of the divided data stream is calculated, as shown in Formula (21).

$$H_{IAT}(\{IP_{src}, IP_{dst}\}) = \sum_j H_j(\{IP_{src}, IP_{dst}\}) = \sum_j (-\sum_i p(MT_i) \log p(MT_i)) \tag{21}$$

$H_j(\{IP_{src}, IP_{dst}\})$ is the information entropy of each segment in the network traffic, and the corresponding information gain is the following:

$$Info_G(\{IP_{src}, IP_{dst}\}) = H(M) - H_{IAT}(\{IP_{src}, IP_{dst}\}) \tag{22}$$

To compare with the division of time intervals, the number of fragments divided by the two methods should be as close as possible. If the number of fragments divided by IAT is N_{IAT} , the corresponding time interval is $\frac{N}{N_{IAT}}$. The information entropy of the corresponding data stream divided by the time interval is shown in Equation (23):

$$H(\frac{N}{N_{IAT}}) = - \sum_{\Delta t_i \in \frac{N}{N_{IAT}}} p(\Delta t_i) \log p(\Delta t_i) \tag{23}$$

The information gain divided by the time interval is shown in Equation (24):

$$Info_G(T) = H(M) - H(\frac{N}{N_{IAT}}) \tag{24}$$

The information gains of the two methods are calculated according to the above methods, and the results are shown in Table 11. To compare with the effect of time division, we use the same number of fragments for the same dataset.

Table 11. Dataset.

Name	Packet Number	Application Fingerprint Number	Total Information Entropy of Data	Time Division	IAT	Information Gain (Time, IAT)	
Dataset1	1755	441	3.24	2.64	1.39	0.60	1.85
Dataset2	115,106	6830	5.06	3.83	1.62	1.23	3.44
Dataset3	139,193	9104	5.14	3.95	1.93	1.19	3.22
Dataset4	73,912	7643	4.87	3.88	1.74	0.99	3.13
Dataset5	76,269	6986	4.88	3.84	1.34	1.04	3.54
Dataset6	68,537	7366	4.84	3.87	1.45	0.97	3.39
Dataset7	102,846	13,857	5.01	4.14	1.43	0.87	3.58
Dataset8	182,292	149,683	5.26	5.18	1.14	0.08	4.12
Dataset9	82,354	7395	4.92	3.86	1.76	1.06	3.16
Dataset10	28,884	3735	4.46	4.07	1.46	0.39	3.00

It can be seen from Table 11 that the information gain of data stream division by IAT is higher than that by time division. The reason is that the data stream divided by time has a lot of contingency, the repeatability of the divided slice is very small, the entropy of the divided information is still large, and the nodes are not pure. Because the “breakpoint” of the data stream can be found accurately by IAT, the repeatability of the application fingerprint is very high. There are only dozens of application fingerprints extracted from potentially large data streams. When the data volume is large enough, more applications are used, and the switching between addresses is frequent. The shorter the fingerprints that are obtained, the more likely they are to be repeated, and the purer the node’s information is, so the information gained will be greater. This indicates that the application fingerprint classification effect is better.

5.3. Fingerprinting Integrity Experiment

Network packets are mostly encrypted data. Due to the huge amount of data, manual marking can only be used as part of the work, mainly through the text clustering of DNS messages to achieve data marking. In the experiment, an application is continuously used to compare with existing DNS messages to determine the type of data. Figure 14 shows the data packets when WeChat started.

1867	82.475.521	182.254.21.82	172.20.10.3	HTTP/X...	572 HTTP/1.0 200 OK
1868	82.475.523	182.254.21.82	172.20.10.3	TCP	54 443 → 1820 [FIN, ACK] Seq = 3307 Ack = 355 win = 15,488 Len = 0
1869	82.475.824	172.20.10.3	182.254.21.82	TCP	54 1820 → 443 [ACK] Seq = 355 Ack = 3308 Win = 16,896 Len = 0
1870	82.510.669	172.20.10.3	182.254.21.82	HTTP	54 GET /cgi-bin/micromsg-bin/newgetdns?uin = 0&clientversion = 1644,495,560&scene = 0&et = 1&md5 = xxxxxxxxxxxxxxxxxxxxxxxx...
1871	82.517.536	172.20.10.3	172.20.10.1	DNS	82 standard query 0x233a A extshort.weixin.qq.com
1872	82.536.632	182.254.21.82	172.20.10.3	TCP	54 443 → 1820 [ACK] Seq = 3308 Ack = 356 win = 15,488 Len = 0
1873	82.551.447	172.20.10.1	172.20.10.3	DNS	198 Standard query response 0x233a A extshort.weixin.qq.com CNAME short.weixin.qq.com A 120.204.10.103 A 183.
1874	82.555.335	172.20.10.3	120.204.0.150	TCP	66 1821 → 80 [SYN, ACK] Seq = 0 Win = 17,520 Len = 0 MSS = 1460 WS = 256 SACK_PERM = 1
1875	82.618.579	120.204.0.150	172.20.10.3	TCP	66 80 → 1821 [SYN, ACK] Seq = 0 Ack = 1 win = 14,400 Len = 0 MSS = 1394 SACK_PERM = 1 WS = 128
1876	82.619.001	172.20.10.3	120.204.0.150	TCP	54 1821 → 80 [ACK] Seq = 1 Ack = 1 win = 17,408 Len = 0
1877	82.634.369	172.20.10.3	120.204.0.150	HTTP	524 POST /mmtls/00,003,271 HTTP/1.1 (application/octet-stream)
1878	82.698.588	120.204.0.150	172.20.10.3	TCP	54 80 → 1821 [ACK] Seq = 1 Ack = 471 Win = 15,488 Len = 0
1879	82.698.589	120.204.0.150	172.20.10.3	HTTP	738 HTTP/1.1 200 OK (application/octet-stream)
1880	82.698.590	120.204.0.150	172.20.10.3	TCP	54 80 → 1821 [FIN, ACK] Seq = 685 Ack = 471 win = 15,488 Len = 0
1881	82.698.843	172.20.10.3	120.204.0.150	TCP	54 1821 → 80 [ACK] Seq = 471 Ack = 686 Win = 16,640 Len = 0
1882	82.708.376	172.20.10.3	120.204.0.150	TCP	54 1821 → 80 [FIN, ACK] Seq = 471 Ack = 686 win = 16,640 Len = 0
1883	82.710.011	172.20.10.3	120.204.0.150	TCP	66 1822 → 80 [SYN] Seq = 0 Win = 17,520 Len = 0 MSS = 1460 WS = 256 SACK_PERM = 1
1884	82.758.848	120.204.0.150	172.20.10.3	TCP	54 80 → 1821 [ACK] Seq = 686 Ack = 472 Win = 15,488 Len = 0
1885	82.763.380	120.204.0.150	172.20.10.3	TCP	66 80 → 1822 [SYN, ACK] Seq = 0 Ack = 1 win = 14,400 Len = 0 MSS = 1394 SACK_PERM = 1 WS = 128
1886	82.763.705	172.20.10.3	120.204.0.150	TCP	54 1822 → 80 [ACK] Seq = 1 Ack = 1 Win = 17,408 Len = 0
1887	82.765.954	172.20.10.3	120.204.0.150	HTTP	1161 POST /mmtls/00,003,271 HTTP/1.1 (application/octet-stream)
1888	82.823.834	120.204.0.150	172.20.10.3	TCP	54 80 → 1822 [ACK] Seq = 1 Ack = 1108 Win = 16,640 Len = 0

Figure 14. The packets of Wechat.

This part of the experiment is to verify the accuracy of fingerprinting extraction, mainly to analyze the comparison between the divided data stream fragment and the standard application fingerprint. The experiment needs to be manually marked first because the amount of data is huge, and the marking cannot be completely accurate. Therefore, through the observation and analysis of the above dataset many times, typical application fingerprints of various application categories are analyzed. Table 12 lists the partial fingerprints of various applications with the highest frequency.

Table 12 shows some application fingerprints obtained by manual analysis. For example, the application fingerprints shown in lines 13–18 show the data packets captured by users using video applications. The network device has a limit on the size of data packets when sending data, and if the size exceeds the limit, they are divided into several packets and sent in sequence. A video will be divided into different types of packets, TCP or UDP, by network devices. This puts forward two evaluation criteria to analyze the address below division after obtaining the effect of the segments of data flow. The first standard is a fragment in the application fingerprints in artificial analysis to obtain the proportion. In the evaluation criteria as shown in (25), the analysis of the evaluation standard is divided into good segments of data flow and the relationship between the real application fingerprinting.

$$DS_FP = \frac{DS \cap FP_s}{FP_s} \tag{25}$$

DS refers to the number of data flow fragments after the repetition is removed, and FP refers to the number of application fingerprints obtained by manual analysis. Figure 15 shows the division of data flows in each dataset.

Table 12. Fingerprinting type.

Number	Fingerprinting
1.	23 : 5, 23 : 6, 23 : 8, 11 : 02, 23 : 3
2.	12 :, 11 : 02, 12 :, 12 :, 12 :
3.	11 : 02, 23 : 3, 11 : 02, 23 : 7, 23.8
4.	23 : 10, 21 :, 23 :, 12 :
5.	23 : 10, 21 :, 23 :, 12 :, 11 : 01, 11 : 03, 11 : 02, 23 : 2
6.	23 : 5, 23 : 8, 23 : 3
7.	23 : 3, 11 : 02, 11 : 02, 24 :
8.	23 : 6, 23 : 8, 11 : 02, 23 : 3, 23 : 5
9.	23 : 5, 23 : 6, 23 : 8, 23 : 3
10.	21 :, 23 : 12, 24 :, 11 : 02
11.	21 :, 23 : 12, 11 : 02, 24 :, 24 :
12.	11 : 02, 23 : 3, 11 : 02, 23 : 5, 23 : 6, 23 : 8
13.	31 :, (11 : 02) _n
14.	24 :, (11 : 02) _n
15.	23 : 3, 21 :, 23 : 16, (11 : 02) _n
16.	31 :, (12 :) _n
17.	(12 :) _{n'} , (11 : 02) _n
18.	23 : 10, 21 :, 23 : 12, (11 : 02) _n

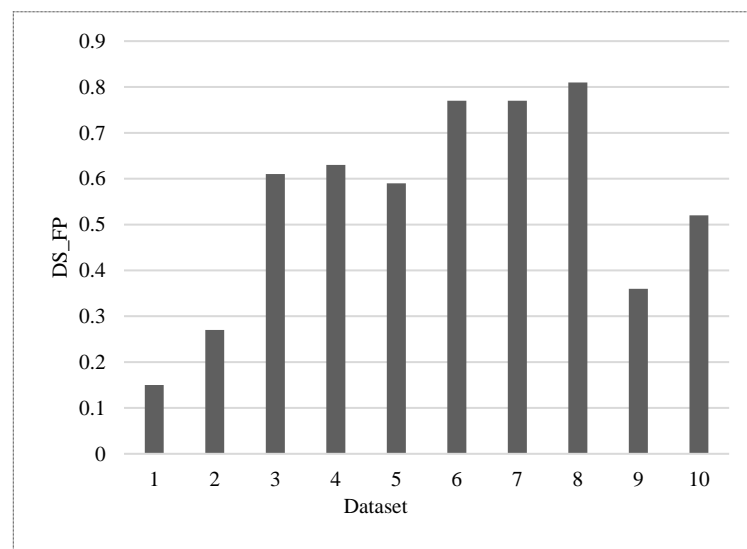


Figure 15. DS_FP comparison.

It can be seen from the analysis in Figure 15 that when the number of packets is small or the application type is single, the segmented data flow fragment type is small. When the dataset is mixed data, the more traffic that is captured, the closer the fingerprint obtained by manual analysis is. The next step is to analyze the part of the grab fragment that does not overlap with the application fingerprint, and (26) is used as the evaluation criterion:

$$FP_{DS} = \frac{\sum_i FP_i}{DS} \tag{26}$$

$\sum_i FP_i$ is the proportion of various application fingerprints in the divided data segments. This evaluation index analyzes the situation of data flow segments that are different from application fingerprints in the traffic. This paper defines such data flow segments that are not application fingerprints as “fragments” of the data stream, aiming to find out the reasons for sending these irregular packets. The analysis of these packet fragments that do not belong to the fingerprint can be of great help to the preprocessing of the data stream

and can remove a lot of interference and noise. Figure 16 shows the “fragments” of the data stream after the traffic division of each data set.

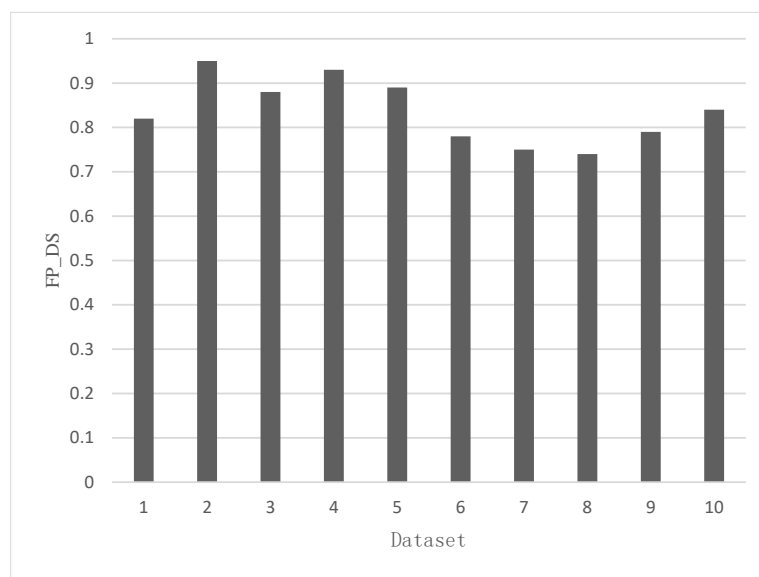


Figure 16. FP_DS comparison.

As shown in Figure 16, a relatively high FP_DS scenario is using a single category of network traffic. In this case, network devices send data packets of the same application in a centralized manner, and data packets are retransmitted and rarely out of order. The reason why TCP data packets are retransmitted is that TCP data packets are retransmitted. The cause of Dataset6, Dataset7, and Dataset8 fragments is that multiple applications send a large number of data packets at the same time, and network devices send data packets to different IP addresses. As a result, there are many fragments with only one or two data packets. Another reason for fragments is the jump of IP addresses. Due to the problems of the server and network structure, different resources will be distributed on different servers, so even a communication session within a period will be sent by different IP addresses, which also causes the data stream to be divided into short fragments.

5.4. High-Utility Fingerprinting Extraction Experiment

This experiment mainly analyzes the application of fingerprinting extraction based on utility. Firstly, the utility of different message types in different datasets is analyzed, and the influence of data stream size and class on the evaluation of message utility is analyzed. Then, the high-utility application fingerprint extracted by the Huf algorithm is analyzed. In this paper, the tf-idf algorithm is used to evaluate the utility of the application. Figure 17 shows tf-idf values of the main types of packets under different datasets.

Figure 17 shows that the category and number of packets have a great influence on the utility of packets. When the number of packets obtained is large enough, the packets tend to be stable. This is similar to natural language processing, which requires a large amount of text to form a huge corpus to ensure the accuracy of the calculation.

The following is the analysis of the Huf algorithm to automatically obtain the effect of application fingerprinting. The Huf algorithm aims can take the place of manual analysis based on the application fingerprints of the session from the network traffic. First of all, obtaining the fingerprint must be accurate, and the requirement is to obtain as much as possible the application fingerprints that contain artificial analysis results. The second step is to reserve a small number of application fingerprints. Compared with TCP and UDP, encrypted messages account for only a small number of network data flows, because they are the part where the client establishes a connection with the server, and the server will maintain the connection for a long time in the future. Therefore, another criterion for

evaluating the Huf algorithm is the retention of these important few application fingerprints. Finally, the last step is to obtain the application of the fingerprint in the manual analysis that does not include the sample. The error extract fingerprint should be as far away as possible; otherwise, it will cause a certain influence on future traffic classification. This paper used the three standard evaluations of the above three kinds of situations. One first needs to access the application of fingerprint artificial analysis, which can be divided into two parts: FP_e that contains the encrypted message and FP_u that does not contain the encrypted message. The evaluation index is shown in (27).

$$Metric_{Huf} = \frac{FP_{Huf}^c}{FP_e + FP_n} \times \frac{FP_{Huf}^e}{FP_e} \times \frac{FP_{Huf}}{FP_{Huf} - FP_{Huf}^c} \tag{27}$$

FP_{Huf} , FP_{Huf}^c , and FP_{Huf}^e are the total number of application fingerprints, the number of correct classifications, and the number of encrypted messages obtained according to the Huf algorithm, respectively. The higher the value of $Metric_{Huf}$ is, the better the effect of the Huf algorithm is. The experiment is conducted on the above dataset, and the results are shown in Figure 18.

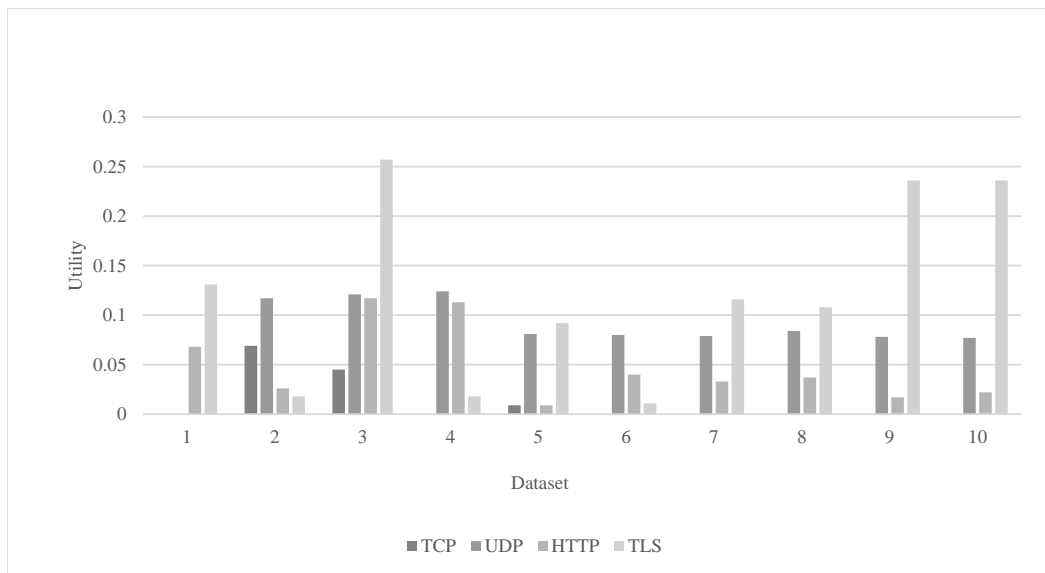


Figure 17. Utility of Messages.

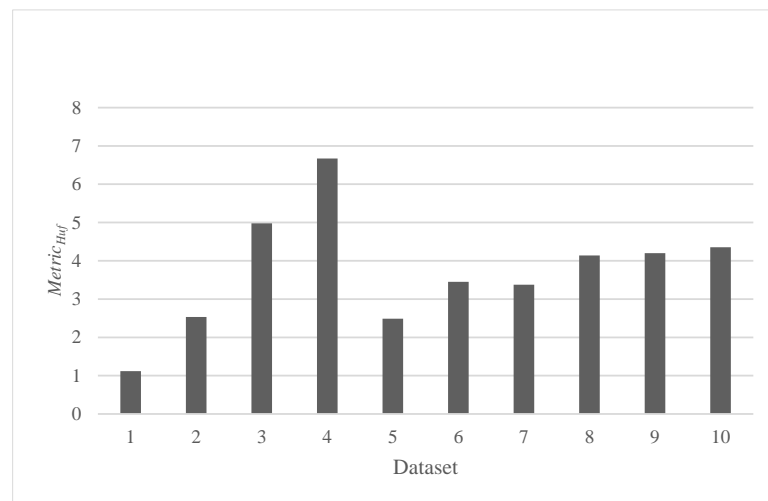


Figure 18. Metric_{Huf} Comparison.

It can be seen from Figure 18 that when the Dataset1–Dataset5 obtained in the experiment have few data packets or the application is a single category, $Metric_{Huf}$ is unstable and irregular. When the data packets collected are large and the time is long, $Metric_{Huf}$ will be stable within a certain range.

6. Conclusions

Extracting network traffic features is always the focus of network behavior analysis, and modeling a large number of independent data packets is the premise of traffic processing. The Huf algorithm proposed in this paper can realize dynamic application fingerprinting extraction, which solves the two problems that anomaly detection cannot process original network datasets and relies on manual extraction of features, and realizes dynamic data flow processing. The application fingerprint extracted by the Huf algorithm has two functions in anomaly detection. The first function is to provide features for traffic classification in the next stage, and the second function is to establish a profile model in anomaly detection together with the fingerprint and classified network traffic. In this paper, the original network data are taken as the analysis object to achieve fine-grained anomaly detection of mobile terminals. The application fingerprint extracted in this paper is the basis of anomaly detection modeling.

In this paper, we use the association analysis algorithm in data mining to mine the relationship between independent data packets. We built a Huf-Tree to realize dynamic application fingerprint mining and calculated the utility of fingerprints to obtain more valuable fingerprints. In this paper, we still have some problems. The IAT in this paper is only divided based on IP addresses, and there is a lot of room for improvement. If the data flow is divided accurately, the subsequent algorithm will obtain better results. For fine-grained anomaly detection methods, the analysis of outlier types needs further research.

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

Funding: This paper is supported by the National Key Research and Development Program (No: 2022YFC3330903) and the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing (No: GJJ-22).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Acknowledgments: The authors sincerely thank the anonymous reviewers for their valuable and constructive comments.

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

References

1. Hajisalem, V.; Babaie, S. A hybrid intrusion detection system based on ABC-AFS algorithm for misuse and anomaly detection. *Comput. Netw.* **2018**, *136*, 37–50. [\[CrossRef\]](#)
2. Taylor, V.F.; Spolaor, R.; Conti, M.; Martinovic, I. Robust Smartphone App Identification Via Encrypted Network Traffic Analysis. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 63–78. [\[CrossRef\]](#)
3. Xu, Y.; Wang, T.; Li, Q.; Gong, Q.; Chen, Y.; Jiang, Y. A multi-tab website fingerprinting attack. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 327–341.
4. Dai, S.; Tongaonkar, A.; Wang, X.; Nucci, A.; Song, D. NetworkProfiler: Towards Automatic Fingerprinting of Android Apps. In Proceedings of the INFOCOM, 2013 Proceedings IEEE, Turin, Italy, 14–19 April 2013; IEEE: Piscataway, NJ, USA, 2013.

5. Khatouni, A.S.; Zincir-Heywood, N. Integrating Machine Learning with Off-the-Shelf Traffic Flow Features for HTTP/HTTPS Traffic Classification. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
6. Friedberg, I.; Skopik, F.; Settanni, G.; Fiedler, R. Combating Advanced Persistent Threats: From Network Event Correlation to Incident Detection. *Comput. Secur.* **2014**, *48*, 35–57. [[CrossRef](#)]
7. Kohout, J.; Pevný, T. Network traffic fingerprinting based on approximated kernel two-sample test. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 788–801. [[CrossRef](#)]
8. Santos, I.; Peña, Y.K.; Devesa, J.; Bringas, P.G. N-Grams-based file signatures for malware detection. In Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Milan, Italy, 6–10 May 2009; pp. 317–320.
9. Kampeas, J.; Cohen, A.; Gurewitz, O. Traffic Classification Based on Zero-Length Packets. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1049–1062. [[CrossRef](#)]
10. Kohout, J.; Komárek, T.; Čech, P.; Bodnar, J.; Lokoč, J. Learning communication patterns for malware discovery in HTTPs data. *Expert Syst. Appl.* **2018**, *101*, 129–142. [[CrossRef](#)]
11. Muehlstein, J.; Zion, Y.; Bahumi, M.; Kirshenboim, I.; Dubin, R.; Dvir, A.; Pele, O. Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser and Application. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 129–142.
12. Shim, K.S.; Ham, J.H.; Sija, B.D.; Kim, M.S. Application traffic classification using payload size sequence signature. *Int. J. Netw. Manag.* **2017**, *27*, e1981. [[CrossRef](#)]
13. Sisodia, D.S.; Khandal, V.; Singhal, R. Fast prediction of web user browsing behaviours using most interesting patterns. *J. Inf. Sci.* **2018**, *44*, 74–90. [[CrossRef](#)]
14. Lu, J.; Gou, G.; Su, M.; Song, D.; Liu, C.; Yang, C.; Guan, Y. GAP-WF: Graph Attention Pooling Network for Fine-grained SSL/TLS Website Fingerprinting. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021.
15. Sun, F.; Zhao, L.; Zhou, B.; Wang, Y. Automatic Fingerprint Extraction of Mobile APP Users in Network Traffic. In Proceedings of the 2020 5th International Conference on Computational Intelligence and Applications (ICCIA), Beijing, China, 19–21 June 2020; pp. 150–155.
16. Almoqbil, R.S.; Rauf, A.; Quradaa, F.H. A Survey of Correlated High Utility Pattern Mining. *IEEE Access* **2021**, *9*, 42786–42800. [[CrossRef](#)]
17. Bao, J.; Zhang, L.; Han, B. Collaborative Attention Network with Word and N-Gram Sequences Modeling for Sentiment Classification. In Proceedings of the International Conference on Artificial Neural Networks, Munich, Germany, 17–19 September 2019; Springer: Cham, Switzerland, 2019; pp. 79–92.
18. Duessel, P.; Gehl, C.; Flegel, U.; Dietrich, S.; Meier, M. Detecting zero-day attacks using context-aware anomaly detection at the application-layer. *Int. J. Inf. Secur.* **2016**, *6*, 475–490. [[CrossRef](#)]
19. Wang, S.; Yan, Q.; Chen, Z.; Yang, B.; Zhao, C.; Conti, M. Detecting Android Malware Leveraging Text Semantics of Network Flows. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1096–1109. [[CrossRef](#)]
20. Korczynski, M.; Duda, A. Markov Chain Fingerprinting to Classify Encrypted Traffic. In Proceedings of the 2014 IEEE Conference on Computer Communications—IEEE INFOCOM 2014, Toronto, ON, Canada, 27 April–2 May 2014; pp. 781–789.
21. Zhang, J.; Xiang, Y.; Zhou, W.; Wang, Y. Unsupervised traffic classification using flow statistical properties and IP packet payload. *J. Comput. Syst. Sci.* **2013**, *79*, 573–585. [[CrossRef](#)]
22. Bhatia, A.; Bahugunaa, A.A.; Tiwaria, K.; Haribabua, K.; Vishwakarmab, D. A Survey on Analyzing Encrypted Network Traffic of Mobile Devices. *arXiv* **2020**, arXiv:2006.12352.
23. Cong, Y. Research on Data Association Rules Mining Method Based on Improved Apriori Algorithm. In Proceedings of the 2020 International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE), Bangkok, Thailand, 30 October–1 November 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 373–376.
24. Xu, T.; Xu, J.; Dong, X. Mining High Utility Sequential Patterns Using Multiple Minimum Utility. *Int. J. Pattern Recognit. Artif. Intell.* **2018**, *32*, 1859017. [[CrossRef](#)]
25. Davashi, R.; Nadimi-Shahraki, M.H. EFP-tree: An efficient FP-tree for incremental mining of frequent patterns. *Int. J. Data Min. Model. Manag.* **2019**, *11*, 144–166. [[CrossRef](#)]
26. Baek, Y.; Yun, U.; Kim, H.; Nam, H.; Lee, G.; Yoon, E.; Vo, B.; Lin, J.C.W. Erasable pattern mining based on tree structures with damped window over data streams. *Eng. Appl. Artif. Intell.* **2020**, *94*, 103735. [[CrossRef](#)]
27. Chang, Y.I.; Li, C.E.; Chou, T.J.; Yen, C.Y. A weight-order-based lattice algorithm for mining maximal weighted frequent patterns over a data stream sliding window. In Proceedings of the 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, Japan, 13–17 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 961–964.
28. Kim, D.; Yun, U. Efficient algorithm for mining high average-utility itemsets in incremental transaction databases. *Appl. Intell.* **2017**, *47*, 114–131. [[CrossRef](#)]
29. Zihayat, M.; Chen, Y.; An, A. Memory-adaptive high utility sequential pattern mining over data streams. *Mach. Learn.* **2017**, *106*, 799–836. [[CrossRef](#)]
30. Ganesan, M.; Shankar, S. High utility fuzzy product mining (HUFPM) using investigation of HUWAS approach. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *13*, 3271–3281. [[CrossRef](#)]