Computers, Materials & Continua
DOI:10.32604/cmc.2021.016562
Article

Tech Science Press

EDSM-Based Binary Protocol State Machine Reversing

Shen Wang[1,*], Fanghui Sun[1], Hongli Zhang[1], Dongyang Zhan[1,2], Shuang Li[3] and Jun Wang[1]

[1]School of Cyberspace Science, Harbin Institute of Technology, Harbin, 150001, China
[2]The Ohio State University, Columbus, 43202, USA
[3]Guangzhou University, Guangzhou, 510006, China
[*]Corresponding Author: Shen Wang. Email: shen.wang@hit.edu.cn
Received: 05 January 2021; Accepted: 12 July 2021

Abstract: Internet communication protocols define the behavior rules of network components when they communicate with each other. With the continuous development of network technologies, many private or unknown network protocols are emerging in endlessly various network environments. Herein, relevant protocol specifications become difficult or unavailable to translate in many situations such as network security management and intrusion detection. Although protocol reverse engineering is being investigated in recent years to perform reverse analysis on the specifications of unknown protocols, most existing methods have proven to be time-consuming with limited efficiency, especially when applied on unknown protocol state machines. This paper proposes a state merging algorithm based on EDSM (Evidence-Driven State Merging) to infer the transition rules of unknown protocols in form of state machines with high efficiency. Compared with another classical state machine inferring method based on Exbar algorithm, the experiment results demonstrate that our proposed method could run faster, especially when dealing with massive training data sets. In addition, this method can also make the state machines have higher similarities with the reference state machines constructed from public specifications.

Keywords: Network security; protocol state machine; EDSM algorithm; protocol reverse engineering; protocol analyzing

1 Introduction

Communication processes among network entities are regulated by network protocols, which define the specifications of message syntaxes and semantics as well as the order messages are to be transmitted. With the continuous development of computer networks [1–3], the mass-produced private or unknown communication protocols have increased the difficulty of related network security management. Thus, the ability of obtaining unknown protocol specifications becomes extremely vital in various situations especially in security-related contexts such as firewalls and intrusion detection systems [4–6]. In these cases, protocol specifications are used to identify malicious traffic and in DPI (Deep Packet Inspection) to assist in making network access more secure and efficient [7–10].

Protocol Reverse Engineering (PRE) refers to the process of analyzing an unknown protocol, where its specifications are obtained by analyzing the message sequences captured from the network traces, or by the instruction streams, during the protocol's communicating process. Network protocol reverse engineering mainly contains two parts:1) message format & semantic mining and 2) state machine inferring. Much research has been done on the first part as represented by the methods in [11–17]. Cui et al. [15] proposed Discoverer to analyze the protocol format in communication traffic. In order to determine the optimal length of message keywords and recover message formats, Cai et al. [16] introduced a hidden semi-Markov model to fit unknown protocol message format. However, there is less research work on the second part [18–24]. Shevertalov presented a protocol state machine inferring solution called PEXT based on common sequences [23]. In order to make the state merging process more reasonable, Comparetti et al. [24] used Exbar algorithm to complete this process and got final minimal DFAs (Deterministic Finite-state Automatons).

In the past years, traditional methods have proven to be usually time-consuming and to contain many errors. Open-source project Samba spent 12 years on manually mining Microsoft's SMB (Server Message Block) protocol specification and implementing cross-platform file and print sharing mechanisms. Afterwards, many automatic protocol reverses analyzing methods were proposed to improve the PRE's efficiency and accuracy, just like the aforementioned research work [18–24]. When reconstructing the state machine of an unknown protocol communication process, however, there has still been many problems with computation efficiency including state explosion and time cost [25,26].

This paper proposes a state merging algorithm based on EDSM [27] to automatically reconstruct the state machine of unknown binary protocols. Firstly, in order to transform protocol message exchange sessions from the form of message sequences into message type sequences, training samples pre-processing is carried out. Secondly, we use a heuristic state labeling algorithm to assign different labels to the protocol transition states. Finally, a state merging algorithm based on EDSM is proposed to complete the state machine reverse process to achieve the final state merging result as a general minimal DFA.

This paper is organized as follows. We give formalized description of the protocol state machine and other definitions in Section 2, and explain the detailed theoretical design of this method in Section 3. In Section 4, several binary protocols are tested by our algorithm. The results demonstrate that the inferred protocol state machines have superior similarities with the reference state machines constructed by public specifications and have better processing capabilities aimed at large amounts of training data.

2 Problem Definition

Protocols regulate the transmission processes among network entities by defining message syntax and semantics as well as their exchange orders. A network protocol consists of three elements: (1) Syntax, (2) Semantic, and (3) Timing. Syntax refers to the message format and encoding. Semantic refers to the meaning of each field in a message. Timing rules the constraints and requirements of communication sequences and state transitions between protocol entities. Specifically, network entities generate responding messages and send them to each other in process of communication according to the semantic information that exists in message format specification and protocol state machines.

Bhargavan et al. [28] formalized the problem of message interaction between network entities as a problem of language identification. Inspired by their work, we give a formalized description of the problem of protocol specification mining, as shown below.

Definition 1: A Protocol State Machine can be represented as a "six-tuple" FSM (Finite State Machine), as shown in Eq. (1):

$$\mathrm{FSM} = (\mathrm{Q}, \mathrm{I}, \mathrm{O}, \sigma, \lambda, q_0) \tag{1}$$

where $\mathrm{Q} = \{q_0, q_1, \ldots, q_k\}$ is a finite set of states; $\mathrm{I} = \{i_1, i_2, \ldots, i_m\}$ is an input set of message formats; $\mathrm{O} = \{o_1, o_2, \ldots, o_n\}$ is an output set of message formats; $\delta\colon \mathrm{Q} \times \mathrm{I} \to \mathrm{Q}$ is the state transition function; $\lambda\colon \mathrm{Q} \times \mathrm{I} \to \mathrm{O}$ is the output function; q_0 is the initial state.

In particular, there are two differences between the FSM in Definition 1 and the classical automatic mechanism. Firstly, for the FSM in Definition 1, all states are acceptable without rejection states. Secondly, any prefix of a valid message sequence is acceptable, and any prefix of an invalid message sequence is acceptable except the last message.

Definition 2: The communication between protocol entity D_1 and protocol entity D_2 is composed of a series of message sequences $T_1, \ldots, T_n$. Each message sequence $T_i = \{p_1 p_2 \ldots p_{|T_i|}\}$ is composed of a certain number of messages, where p_i refers to the i^{th} message, $|T_i|$ refers to the number of messages in T_i.

According to Definition 2, we divide the PRE-process into two steps based on the scope of protocol specification: (1) message format and semantic mining, and (2) protocol state machine reverse. Besides, the first step is the basic of the second step. In this paper, we mainly focus on protocol state machine reverse.

3 The Proposed Protocol State Machine Reverse Method

In last section, we give the formalized description of the protocol state machines and explain the relationship between the message format, semantic mining and protocol state machine reverse.

In this section, we introduce the theoretical method design of our proposal. In Section 3.1, we describe how to pre-process the initial message of binary protocols in detail. In Section 3.2, the state labeling algorithm is described to assign different labels on states. Finally, the EDSM-based state merging algorithm is introduced to rebuild final protocol state machines of the target unknown protocol.

3.1 Pre-Processing of Messages

As the initial messages cannot be used as elements of the input set in Definition 1, we preprocess training samples to find state relevant fields to transform protocol sessions from the form of message sequences into the form of message type sequences. The main idea is using multiple sequence alignments to identify the variable length fields in messages, and then extracting state relevant fields by analyzing statistical characteristics of each field after removing variable length fields.

3.1.1 Identification of Variable Length Fields

Binary protocol messages are formed by a series of bytes, which usually consist of message headers and message bodies. The message header includes the message type, length and other fields. The length of a message header is usually fixed. However, in some complex binary

protocols, there may be variable length fields, such as parameter fields. Therefore, we use multiple sequences alignment [29] to identify these variable length fields.

Multiple sequence alignment is essentially an NP-complete problem, which is always realized by heuristic algorithms. Here, we use a progressive multiple sequence alignment algorithm to solve the multiple message field segmentation problem. To implement a multiple sequence alignment, we create a phylogenetic tree to guide this process.

Once the phylogenetic tree is completed, a progressive multiple sequences algorithm can be achieved under the guidance of this phylogenetic tree. In fact, the basic operation of the progressive multiple sequence algorithm is global sequence alignment. In this paper, we apply the Needleman–Wunsch algorithm [30] to complete the global sequence alignment. The score matrix of messages A and B in the Needleman–Wunsch algorithm can be built based on the following equation.

$$\mathrm{S(i,j)} = \max \begin{cases} S(i-1,j-1) + c(A_i, B_j) \\ S(i-1,j) + W_{k\_column} \\ S(i,j-1) + W_{k\_row} \end{cases}, \tag{2}$$

where $1 \le \mathrm{i} \le \mathrm{M},\ 1 \le \mathrm{j} \le \mathrm{M}$.

In order to ensure that the fixed fields do not introduce gaps, and variable length fields could be aligned by adding gaps, we present a new substitution matrix for Needleman–Wunsch algorithm: initialize a match score of 2, mismatch score of -1, gap penalty of -3. The gap penalty is a special treatment in the algorithm running process: continuous gaps are counted only once, and gap penalty plus -1 for the next time.

The advantage of our method is that once gaps are introduced between fixed fields, the remaining fields will not be aligned to result in serious gap penalties. In addition, variable length fields may need to be introduced more than one gap to be aligned. In order to avoid excessive gap penalties, we only count continuous gaps once.

3.1.2 Extraction of State Relevant Fields

There are various fields in binary message formats with different constraints added by protocol specification, which leads to different statistical characteristics of fields. Thus, we compute and analyze their statistical features, and then find out the features we are interested in. Based on the "Variance of the Distribution of the Variances" [31] of fields in each message format, relevant features are filtered for subsequent analysis. In this paper, we make the following assumptions to find the fields related to the state.

Assumption 1. Binary protocols have the following properties:

(1) The specific logic underlies in different traffic flows of a certain protocol to ensure the stable running of transmissions;

(2) State relevant fields in messages assign the common logic of protocols;

(3) The value of state relevant fields is usually limited and not excessive;

(4) The value distributions of state relevant fields are similar in each session;

(5) The length of a state relevant field is 1 byte which can represent 256 message types.

We define the byte as the basic field unit. Based on these assumptions, we can find that there exists a certain change pattern of each byte field in the flow of a protocol. Relevant byte fields in different packets are presented in Fig. 1. The distribution of one field in different flows is similar.

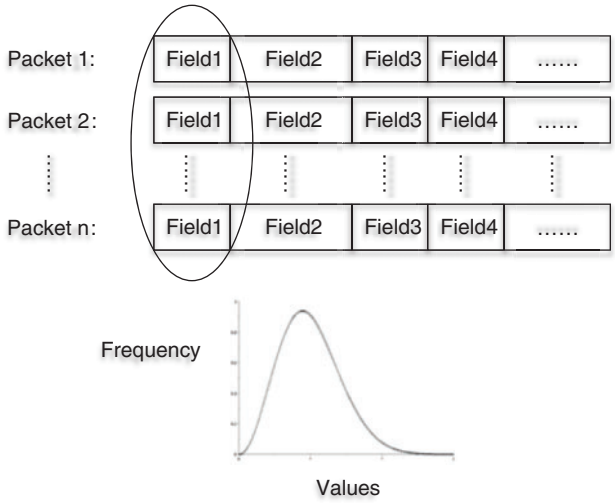

Figure 1: Relevant byte fields in different packets

For comparison, we use $\delta^2_{i,j}$ to denote the decentralization of field i in flow j. Then, calculate the variance distribution of each field in the entire set of flows 1...n. The distribution curve shows the variability of the field in these flows. Then, we consider in all flows 1...n for each field of the variance distribution previously computed $\delta^2\{\delta^2_{i,1}, \delta^2_{i,2}, \ldots, \delta^2_{i,n}\}$. What we are interested in is the low degree of variability in this statistic.

3.2 EDSM-Based Protocol State Machine Reverse

We determine state relevant fields by preprocessing. At the end of the preprocessing, each session is denoted as a sequence $S_i = (t_1, \ldots, t_n)$, and $t_1, t_2, \ldots, t_n$ represents the message type set. In this process of state machine inference, we are aiming to achieve an acceptor machine which can recognize the target protocol in its valid sessions by analyzing the sequence of message types.

3.2.1 State Labeling Algorithm

Here, we use an Augmented Prefix Tree Acceptor (APTA [32]) T to build the initial state machine. In the training set, all states are assigned by "accept". Now, we cannot leverage the existing state merging algorithm to merge pairs of states directly, which would result in an overgeneralized DFA with only one single state. Therefore, we introduce and optimize a state labeling algorithm proposed in [24] to assign different labels to the states of T. In the following, we first introduce the state labeling algorithm, and then optimize it to complete state labeling.

There is a convention in network protocols that a sequence of messages must be sent before the server can execute certain actions. So, a regular expression shown in Eq. (3) is used to represent the prerequisite of a certain message.

$$\mathrm{P_m} = .*\mathrm{r}(a_1||a_j) \tag{3}$$

where r, $a_1, \ldots, a_j$ are message types.

A prerequisite means the server in a state that can accept message of type m must receive another type r firstly, and follows with $(a_1|\ldots|a_j)^*$, optionally.

The limitation of an algorithm for calculating r is that more than one value of r may be obtained after the calculation, but not each value is reasonable. For example, in SMB, the "OPEN" operation must be performed before the "WRITE" operation, and the "TREE CONN" operation must be performed before the "OPEN" operation. Therefore, we can conclude that the "OPEN" operation is likely to rely on the "TREE CONN" caused by the dependent transition.

Therefore, for each value r_i, we test whether r_i is the last appeared after other values in each protocol session. If such a value is found, we consider it as the final value r that we required. And if we cannot find one, the value of r is null which means no message type is required before the server can accept the message type M.

Once all prerequisites are computed, the state q of T will be labelled within the set of message types, which can serve as an input. A serious problem of the state labeling algorithm is that many message types have the same prerequisites, which results in many states having the same label, but not all of them actually should be merged. For example, in SMB, the state created by "OPEN"' operation should not be merged with the state created by the "CREATEDIR" operation. The reason is that the "WRITE" and "READ" operations have relied on the "OPEN" operation, but no operation relies on the "CREATEDIR" operation. So, we label the state "s" by a tuple:

$$\text{Label} = (\text{A}, \text{R}, \text{O}) \tag{4}$$

where A is an acceptable message type of s, R is a set of message types that require s, and O is the last two elements of the path from the root to s.

The expanded state labeling algorithm is shown as follows.

Algorithm 1: Expanded_State_Labeling

```
Input: An APTA T
Result: the set of state label L
Path = φ, A = φ, R = φ, O = φ, o = null
for each state s in T do
        Path = message type sequences from root to s in T
        O = last two elements in Path
        o = last element in Path
for each message type t do
        if Path satisfy the prerequisite of t
                add t to A
for each message type t do
        if t requires o
                add t to R
add ⟨s, ⟨A, R, O⟩⟩ to L
return L
```

3.2.2 EDSM-Based State Merging Algorithm

An essential operation of protocol state machine reverse is the similar states merging. According to the state tree (APTA [32]) labelled by heuristics, now we can go one step further to infer an optimal DFA by merging similar states. In the field of grammar inference, obtaining the smallest

DFA consistent with the labelled training set has been proved to be an NP-complete problem by Gold, and there are many exact or approximate algorithms to solve this problem.

In fact, the complexity of protocol state machine inference is always higher than general grammar inference, because the amount of protocol messages is larger and the protocol logic is more complex. Therefore, in this paper, we adopt an approximate algorithm, EDSM, to deal with the state merging of unknown protocol APTAs. The performances of EDSM [33] and Exbar are compared and analyzed in Section 4.3.

The EDSM algorithm is achieved in the red-blue frame, which is a directed graph with the following properties:

(1) All nodes in the graph are labelled with red, blue or unmarked;

(2) The initial root node is marked with red, and its children are marked with blue;

(3) Each red node's non-red children are marked with blue;

(4) Each unmarked blue node is the root of a tree;

EDSM is based on a greedy strategy. It will calculate all scores of red and blue node pairs using state labels. If there exists a blue node that cannot be merged with any other red nodes, we promote it to be red. The red node and blue node with the highest score will be merged. In the process of protocol state machine reversal, we use the Breadth-First Search (BFS) strategy to modify the EDSM algorithm in order to complete the state merging of states in same depth with same behavior. The modified EDSM algorithm is shown in Algorithm 2.

Algorithm 2: EDSM-BFS

```
Input: red nodes of T
blue_node = Φ, target_depth = 0, merges = φ, promotable_blue = φ
Traverse T with breadth first search
    if current node n is blue node
        target_depth = n.depth
        break
Traverse T with breadth first search
    if current node n is blue node and n.depth <= target_depth + 1
        add n to blue_node
merges = sort_by_score (filter (successful, (map (try merging and then undo, cross produce (red nodes, blue nodes))))
promotable_blue = filter (appears in no merge, blue nodes)
if promotable_blue! = Φ
    EDSM-BFS (cons (min-depth (promotable_blue), red nodes)))
else if merges != Φ
    try merging (first (merges))
EDSM-BFS (red nodes)
```

There are two places reflecting the idea of breadth-first search. The first is the selection of blue nodes to generate the candidate state merging set; and the second is the selection of promotable blue nodes. The advantage of using breadth-first search is that it can search similar states in different paths preferentially to avoid generating too many branches in the final state machine.

In order to obtain an efficient and reliable result, the focus of our algorithm is to design a reasonable scoring mechanism. In this paper, we use the state labeling method discussed in Section 3.1 to solve this problem. The scoring algorithm is shown in Algorithm 3.

Algorithm 3: Try_Merging

```
Input red node r, blue node b
Merge score = 0
if label (r.A) = label (b.A) and label (r.R) = label (b.R)
        merge_score + 1
        if label (r.O) = label(b.O) or r.O.second != b.O.second
                merge score + 1
else
        return − 1
for each message type m in M do
        if exist (r.child (m)) and exist (b.child (m))
                Repeat the above operations
return merge_score
```

4 Experiment Results and Analysis

We test our implementation of the proposed method on a number of stateful binary protocols (including transport layer protocol TCP (Transmission Control Protocol) and two application layer protocols SMB and DHCP (Dynamic Host Configuration Protocol)). Since the completeness and validity of the training set have a crucial impact on the quality of the state machine, and the inferred FSM cannot identify the packet not included in the training set, we try to collect as many complete protocol sessions as possible.

4.1 State Machine Inference

In this section, we apply our method to one transport layer protocol (TCP) and two application layer protocols (SMB, DHCP). It creates state machines ranged from 4 to 12 states for each protocol.

TCP. It is an important transport layer protocol which is object oriented, reliable and based on stream of bytes. In our experiments, we connect our terminal with a router, and run Wireshark (a well-known network packet analysis software) to sniff TCP network traffic. Then, the collected traces are fed to the implemented system, and the obtained state machine is shown in Fig. 2.

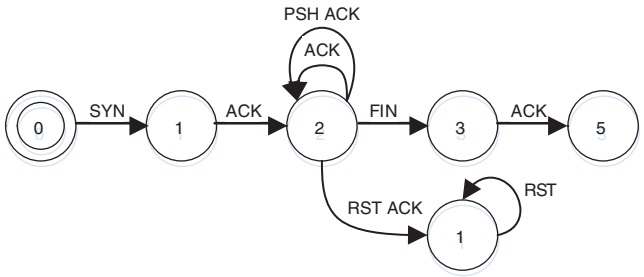

Figure 2: Result of TCP state machine

The three-way handshake of TCP related to State 2 is obviously visible as well as the four waves of TCP leading to State 5. The network data transmit between server and client in State 2. Besides, we can see the "RST" and "RST-ACK" packets in State 4 which are always associated with abnormal connections in the real world.

SMB. As an instance of a relatively elaborate, stateful, binary protocol, we chose SMB to test this method. In the experiment, version 4.1.14 of the Samba software suite has been adopted to tracked the SMB daemon when this client is used to look through directories, carry out typical operations like reading, writing, and deleting directories and files. In this way, a set of 445 recorded sessions is collected. Fig. 4 shows the protocol state machine inferred from this SMB data set.

In Fig. 3, we can see the obvious login sequence leading to State 3. When the "DFS" option is enabled, the client first attaches the "IPC$" share to achieve a "DFS" referral of the requested share. Otherwise, the client immediately enters the requested share of State 6, where most file system operations (including opening, reading, writing, or closing) are available.

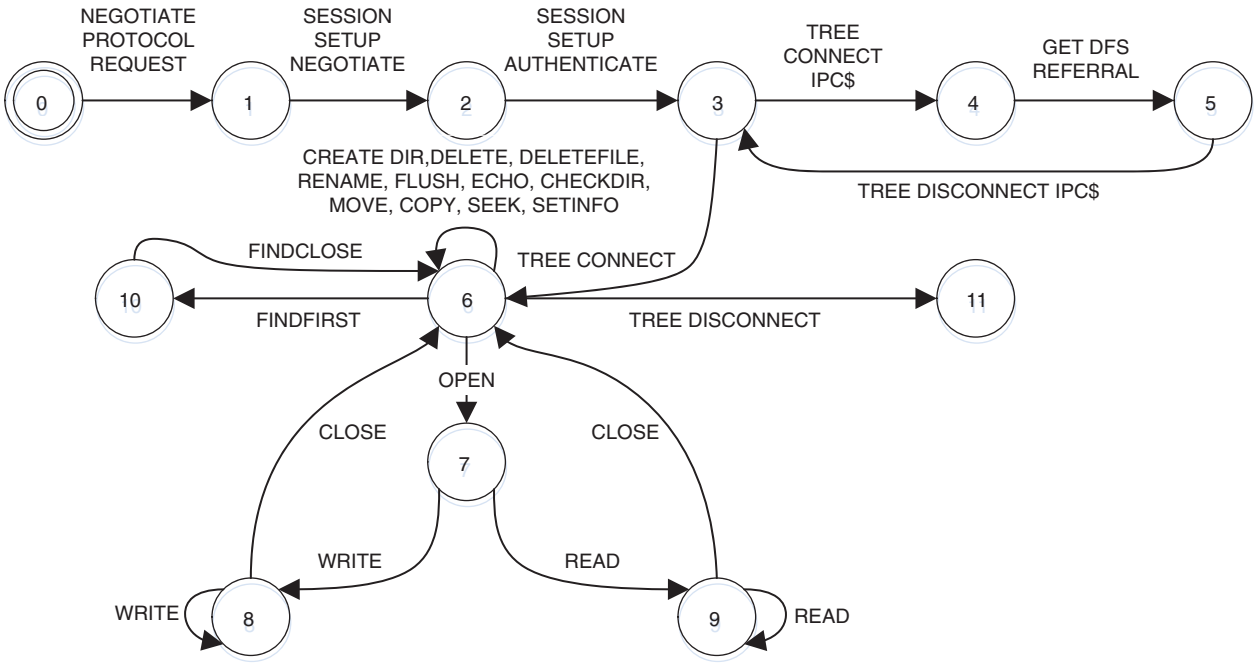

Figure 3: Result of SMB state machine

DHCP. Dynamic host configuration protocol is an important network protocol in the local area network which mainly automatically assigns IP addresses for Internet service providers. In our experiment, we have configured a DHCP server in our local area network and traced the DHCP traffic. 168 DHCP sessions are collected. Fig. 4 shows the DHCP result state machine.

There are two login sequences (0 –> 1 –> 2 and 0 –> 2) leading to State 2 in DHCP state machine. Path 0 –> 1 –> 2 happens when the client logins the server for the first time, and path 0 –> 2 occurs when the client reboots.

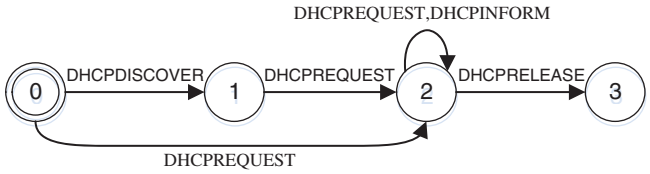

Figure 4: Result of DHCP state machine

4.2 Quality of Protocol State Machine

To test the performance of this system, we use the soundness and completeness to evaluate the quality of the state machines inferred from our implementation.

4.2.1 State Machine Completeness

Generally, we consider a state machine is complete enough if it accepts all valid sessions, and we use the recall rate to measure the completeness of that state machine. There are two methods to obtain test samples: by net trace which reflects the overall completeness of the state machine, or by constructing a reference state machine using protocol specifications which reflects the structural completeness of the state machine. Tabs. 1 and 2 show the results of each method where "IM-" refers to the relative inferred state machines and "RM-" the reference state machines.

Table 1: Overall completeness testing of inferred state machines

No.	State machines	Protocol types	Sessions produced by net-trace	Sessions accepted by IM	Recall
1	IM-TCP	TCP	4377	4377	1
2	IM-SMB	SMB	1120	1093	0.975
3	IM-DHCP	DHCP	540	540	1

Table 2: Structural completeness testing of inferred state machines

No.	Inferred state machines	Reference state machines	Sessions produced by RM	Sessions accepted by IM	Recall
1	IM-TCP	RM-TCP	10000	9540	0.954
2	IM-SMB	RM-SMB	10000	9660	0.966
3	IM-DHCP	IM-DHCP	10000	9669	0.967

From Tabs. 1 and 2, we can see that the recalls of both overall completeness and structural completeness are high enough closing to 100% which demonstrate that our method is useful to accept valid protocol sessions. Besides, overall completeness is a little higher than structural completeness. It is understandable that the test samples produced by the reference state machines are more complete than those by net-traces, and the test samples we used to construct the state machine are collected by net-traces.

4.2.2 State Machine Soundness

We consider a state machine is sound enough if it rejects all invalid protocol sessions, and we use accuracy to measure the soundness of the state machine. The result is shown in Tab. 3, where we can conclude that the state machine we inferred are not over-generalized, which means they can reject invalid sessions.

Table 3: Structural completeness testing for state machine

No.	State machine	Protocol	Sessions created by IM	Sessions tested successfully by RM	Precision
1	RMtcp	TCP	10000	9877	0.988
2	RMsmb	SMB	10000	10000	1
3	RMdhcp	DHCP	10000	10000	1

4.3 Comparative Evaluation

With merely positive examples, there exist other approaches to implement the automation inferring mission. One popular approach is the Exbar algorithm, which is an exact algorithm to infer the minimal consistent DFA. To compare the performance of the proposed method with the Exbar algorithm, we calculate the precision and recall at different numbers of SMB protocol training samples for both two algorithms. The results are shown in Figs. 5 and 6.

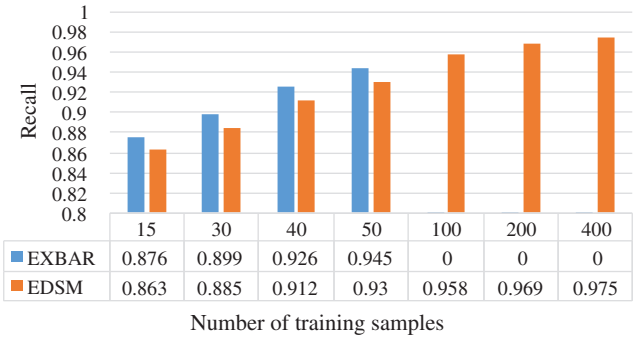

Figure 5: Recall of SMB state machine in different scales of training samples

Two useful conclusions could be drawn from the results.

1) About the impact of the number of training samples. The number of training samples has a relatively large influence on the recall of these two algorithms, and the recall is rising along with the increase of training samples. Exbar performs slightly better than EDSM in recall. By using an improved state labeling algorithm in our system, the accuracy of the two algorithms reaches 100% at the number of different training samples, which avoids merging invalid state pairs. So, the state machine we inferred is not over generalized, and it is unlikely to generate protocol sessions that cannot be accepted by the reference state machine.

2) About the data processing capacity. When the number of training samples exceeds 50, Exbar will not run. EDSM is an approximate algorithm that can ensure inferring state machine in polynomial time. However, Exbar is an exact algorithm using backtracking search strategy, and

tries to find an optimal result with the least states in an almost exhaustive manner. Each time Exbar is called recursively, it only tries to merge a pair of states or promote a blue node to red, and the depth of the search will continue to increase. Once a search fails, Exbar algorithm will return to the last position and choose another search direction, which may easily fall into infinite backtracking and cannot exit. Fig. 7 depicts the search path of Exbar and EDSM when inferring state machines.

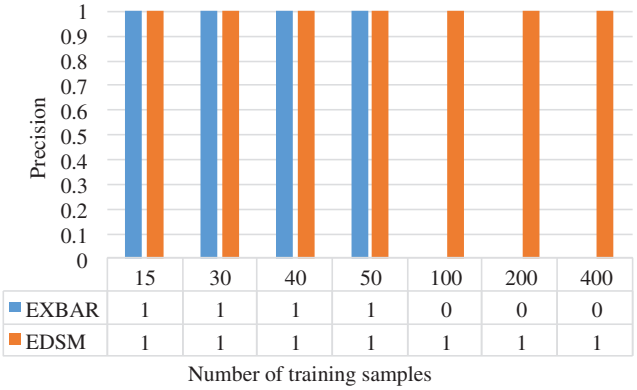

Figure 6: Precision of SMB in different scales of training samples

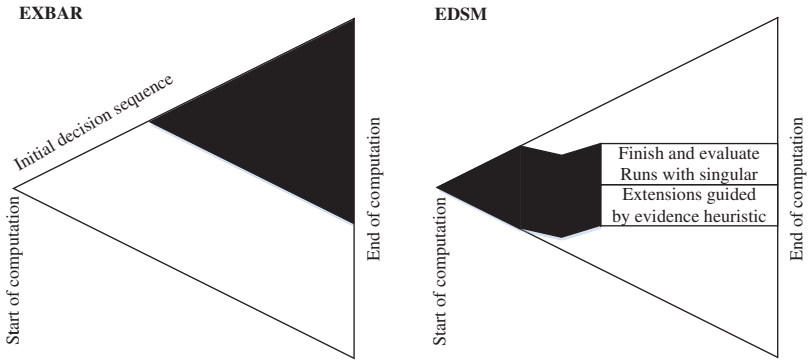

Figure 7: Search direction of Exbar and EDSM

In order to compare the time consumption of the two algorithms more directly, we record their time consumption and search times under different numbers of SMB training samples (see Tab. 4). It can be seen from the results that the running time of the two algorithms grows with the increase of the number of training samples. But for each test set, EDSM takes less time than Exbar. When the sample number in the test set is equal to or greater than 100, Exbar would fail to give a result while EDSM just takes a little longer time to display the results. The reason for this phenomenon can be explained by the difference in computational complexity between Exbar and EDSM. As an approximation algorithm, the time complexity of the EDSM algorithm is much lower than that of Exbar, which makes it more effective when the data size increases. Therefore, the EDSM algorithm has a better performance in the face of large data and higher real-time requirements.

Table 4: Time consuming between Exbar and EDSM

Training sample count		15	30	40	50	100	200	400
Time consuming	Exbar	0.15 s	0.735 s	5.925 s	34.196 s			
	EDSM	0.056 s	0.161 s	0.387 s	0.645 s	2.81 s	3.5 s	4.097 s
Search times	Exbar	228	732	3078	6796			
	EDSM	196	427	688	854	1556	2440	3056

5 Conclusion

This paper proposes a valid approach to refer the state machine of unknown binary protocols from network traces, especially when the training samples are large. Firstly, we improve the substitution matrix of multiple sequence alignments to identify variable length fields and remove them. Then, we extract state relevant fields by analyzing their statistical characteristics. To infer a minimal DFA consistent with training samples, we optimize a state labeling algorithm and apply an optimized EDSM algorithm to complete the final state merge.

In order to validate the method implemented in this paper, we test our system on three typical binary protocols: TCP, SMB and DHCP. The experimental results show that the state machine we inferred is reliable in terms of both completeness and soundness. Compared with the Exbar algorithm, the experiment results show that when Exbar totally fails, our system has better performances in processing a large number of training samples. In some application environments, such as instruction detection, malicious traffic recognition and other network protection mechanisms, the ability to handle with big data provided by EDSM algorithm will be more practical.

In the future, the method for selecting an appropriate algorithm to accomplish the state machine reconstructing needs to be studied in depth. As this article proves, EDSM performs better on large data, but has a slightly lower recall rate than Exbar. We will continue working on this topic and find an automatic mechanism to utilize the proper algorithm to obtain better protocol reverse results. Besides, combined with the present method, the Markov Model could also be considered to deal with the packet missing problem in the future.

Acknowledgement: We would like to express our gratitude to all those who gave supports for work of this paper, and also devote our great thanks to all the anonymous reviewers of this paper, whose precious comments help promote the quality of this paper a lot.

Funding Statement: This work is supported by the National Natural Science Foundation of China (Grant Number: 61471141, 61361166006, 61301099), Basic Research Project of Shenzhen, China (Grant Number: JCYJ20150513151706561), and National Defense Basic Scientific Research Program of China (Grant Number: JCKY2018603B006).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] I. Farris, T. Taleb, Y. Khettab and J. Song, "A survey on emerging SDN and NFV security mechanisms for IoT systems," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 812–837, 2019.

[2] D. E. Kouicem, A. Bouabdallah and H. Lakhlef, "Internet of things security: A top-down survey," *Computer Networks*, vol. 141, no. 4, pp. 199–221, 2018.

[3] C. Chu, Z. Huang, R. Xu, G. Wen and L. Liu, "A cross layer protocol for fast identification of blocked tags in large-scale rfid systems," *Computers, Materials & Continua*, vol. 64, no. 3, pp. 1705–1724, 2020.

[4] H. Choi, M. Kim, G. Lee and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *Journal of Supercomputing*, vol. 75, no. 9, pp. 5597–5621, 2019.

[5] L. Shi, Q. Liu, J. Shao and Y. Cheng, "Distributed localization in wireless sensor networks under denial-of-service attacks," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 493–498, 2021.

[6] X. D. Hao, J. M. Zhou, X. Q. Shen and Y. Yang, "A novel intrusion detection algorithm based on long short term memory network," *Journal of Quantum Computing*, vol. 2, no. 2, pp. 97–104, 2020.

[7] Z. Trabelsi, S. Zeidan and M. M. Masud, "Network packet filtering and deep packet inspection hybrid mechanism for IDS early packet matching," in *Proc. IEEE 30th Int. Conf. on Advanced Information Networking and Applications*, Crans-Montana, Switzerland, pp. 808–815, 2016.

[8] S. Zamfir, T. Balan, F. Sandu and C. Costache, "Solutions for deep packet inspection in industrial communications," in *Proc. Int. Conf. on Communications*, Bucharest, Romania, pp. 153–158, 2016.

[9] B. Yang and D. Liu, "Research on network traffic identification based on machine learning and deep packet inspection," in *Proc. IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conf.*, Chengdu, China, pp. 1887–1891, 2019.

[10] C. Luo, S. Su, Y. Sun, Q. Tan, M. Han *et al.,* "A convolution-based system for malicious URLS detection," *Computers, Materials & Continua*, vol. 62, no. 1, pp. 399–411, 2020.

[11] I. Bermudez, A. Tongaonkar, M. Iliofoto, M. Mellia and M. M. Munafo, "Towards automatic protocol field inference," *Computer Communications*, vol. 84, no. 4, pp. 40–51, 2016.

[12] K. Shim, Y. Goo, M. Lee, H. Hasanova and M. Kim, "Inference of network unknown protocol structure using CSP (contiguous sequence pattern) algorithm based on tree structure," in *Proc. EEE/IFIP Network Operations and Management Symp.*, Taipei, Taiwan, 2018.

[13] G. Lodi, L. Buttyon and T. Holczer, "Message format and field semantics inference for binary protocols using recorded network traffic," in *Proc. 26th Int. Conf. on Software, Telecommunications and Computer Networks*, Split, Croatia, pp. 105–110, 2018.

[14] F. Sun, S. Wang, C. Zhang and H. Zhang, "Unsupervised field segmentation of unknown protocol messages," *Computer Communications*, vol. 146, pp. 121–130, 2019.

[15] W. Cui, J. Kannan and H. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proc. USENIX Security Symp.*, Boston, MA, pp. 199–212, 2007.

[16] J. Cai, J. Z. Luo and F. Lei, "Analyzing network protocols of application layer using hidden semi-markov model," *Mathematical Problems in Engineering*, vol. 4, pp. 1–14, 2016.

[17] F. Sun, S. Wang, C. Zhang and H. Zhang, "Clustering of unknown protocol messages based on format comparison," *Computer Networks*, vol. 179, no. 4, pp. 107296, 2020.

[18] Z. Zhang, Q. Wen and W. Tang, "Mining protocol state machines by interactive grammar inference," in *Proc. Third Int. Conf. on Digital Manufacturing & Automation*, Guilin, China, pp. 524–527, 2012.

[19] G. Bossert, G. Frédéric and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proc. 9th ACM Symp. on Information, Computer and Communications Security*, Kyoto, Japan, pp. 51–62, 2014.

[20] L. Zhao, X. Liang, X. Peng, H. Kong and M. Wang, "An automatic network protocol state machine inference method in protocol reverse engineering," *Applied Mechanics and Materials*, vol. 513–517, pp. 2496–2501, 2014.

[21] M. Xiao and Y. Luo, "Automatic protocol reverse engineering using grammatical inference," *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 5, pp. 3585–3594, 2017.

[22] C. Lee, J. Bae and H. Lee, "PRETT: protocol reverse engineering using binary tokens and network traces," *IFIP Advances in Information and Communication Technology*, vol. 529, pp. 141–155, 2018.

[23] M. Shevertalov and S. Mancoridis, "A reverse engineering tool for extracting protocols of networked applications," in *Proc. 14th Working Conf. on Reverse Engineering*, Vancouver, Canada, pp. 229–238, 2007.

[24] P. M. Comparetti, G. Wondracek, C. Kruegel and E. Kirda, "Prospex: Protocol specification extraction," in *Proc. IEEE Symp. on Security and Privacy*, Berkeley, CA, pp. 110–125, 2009.

[25] C. Tîrnăucă, "A survey of state merging strategies for DFA identification in the limit," *Triangle Language Literature Computation*, vol. 8, pp. 121–136, 2018.

[26] Y. Li, L. Yin, Y. Chen, Z. Yu and N. Wu, "Optimal Petri net supervisor synthesis for forbidden state problems using marking mask," *Informati on Sciences*, vol. 505, no. 10, pp. 183–197, 2019.

[27] K. J. Lang, B. A. Pearlmutter and R. A. Price, "Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm," in *Proc. Int. Colloquium on Grammatical Inference*, Heidelberg, Berlin, pp. 1–12, 2006.

[28] K. Bhargavan, S. Chandra, P. J. Mccann and C. A. Gunter, "What packets may come: Automata for network monitoring," *ACM SIGPLAN Notices*, vol. 36, no. 3, pp. 206–219, 2001.

[29] D. J. Bacon and W. F. Anderson, "Multiple sequence alignment," *Journal of Molecular Biology*, vol. 191, no. 2, pp. 153–161, 1986.

[30] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[31] A. Trifilò, S. Burschka and E. Biersack, "Traffic to protocol reverse engineering," in *IEEE Symp. on Computational Intelligence for Security and Defense Applications*, Ottawa, pp. 1–8, 2009.

[32] M. Bugalho and A. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recognition*, vol. 38, no. 9, pp. 1457–1467, 2005.

[33] K. J. Lang, "Faster algorithms for finding minimal consistent DFAs," NEC Research Institute, Tech. Rep., 1999.