Journal of Network and Computer Applications 171 (2020) 102819

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

ReFSM: Reverse engineering from protocol packet traces to test generation by extended finite state machines

Ying-Dar Lin [a], Yu-Kuen Lai [b,*], Quan Tien Bui [a], Yuan-Cheng Lai [c]

[a] Department of Computer Science, National Chiao Tung University, Hsin-Chu, Taiwan
[b] Department of Electrical Engineering, Chung-Yuan Christian University, Chung-Li, Taiwan
[c] Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan

ARTICLE INFO

Keywords:
EFSM inference
Protocol reverse engineering
Protocol semantic deduction

ABSTRACT

Protocol reverse engineering is helpful to automatically obtain the specifications of protocols that are useful for network management, network security systems and test case generation tools. To achieve better accuracy, these kinds of applications require good models that can capture not only the order of exchanging messages (control flow aspect) but also the data being transmitted (data flow aspect). However, current techniques only focus on inferring the control flow represented as a Finite State Machine (FSM) and without interpreting the data flow. The Extended Finite State Machine (EFSM), embedding memory in the states and data guard in the FSM transitions, is a method commonly used to represent the data flow. In this work, we propose ReFSM, a novel approach to infer the EFSMs of protocols from only network packet traces. The proposed method is evaluated by using datasets of real-world network traffic traces of four protocols: FTP, SMTP, BitTorrent and PPLive. Based on the results, the coverage, accuracy scores of correctness and behavior of inferred models are always higher than 90%. The precision and recall values of message type identification are, at least, well above 94% and 96%, respectively. The inferred EFSMs are close to the correct model derived from protocol specification.

1. Introduction

Detailed understanding of protocol specification is helpful not only for the operation of network intrusion detection systems but also essential for the development of protocol fuzzer and test case generation tools. In the testing domain, smart fuzzers are one of the most useful tools for testing the robustness of the implementation of network security systems. Smart fuzzers need to know how to generate the right messages in the right states (Bossert et al., 2014). Furthermore, based on behavioral models, a set of conformance test cases aimed at verification can also be generated by automated test case generation tools (Tappler et al., 2017; Dahbura et al., 1990).

However, obtaining protocol specifications is a tedious and time-consuming task. For instance, it took more than 12 years to complete the reverse engineering of the SMB protocol of Microsoft Server Message Block (Cui et al., 2007). Even for open protocols such as FTP, HTTP, and MQTT, it takes much time and effort to carefully analyze and manually translate the open source documents to formulate a behavioral model. In addition to such known protocols, there are more than 40% of Internet traffic belong to such unknown application protocols (Wang et al., 2011), and specifications are not available for those proprietary protocols. A considerable number of them are private protocols that enterprises tend to conceal while others are malware and botnets, which are hidden by attackers. Therefore, automatic protocol reverse engineering has recently been proposed to carry out such specification inference.

1.1. Motivation

As indicated in the PI Project (Beddoe, 2018), the methods of protocol reverse engineering can be divided into two types of *execution trace* (application inference) and *network trace* (network inference) (Sija et al., 2018; Kleber et al., 2019; Duchene et al., 2018). Although application inference can formulate a behavioral model very effectively, it is difficult to carry out as a result of the unavailability of specifications and source codes (Kleber et al., 2019).

The network inference method relies on protocol traces to actively reconstruct the behavioral model and infer protocol message formats. Intrusion detection systems typically rely on a parser to perform deep packet inspection based on protocol specifications (Paxson, 1999) (Amiri et al., 2011). The operation of network honeypots, which

* Corresponding author.
E-mail address: ylai@cnsrl.cycu.edu.tw (Y.-K. Lai).

https://doi.org/10.1016/j.jnca.2020.102819
Received 13 February 2020; Received in revised form 31 July 2020; Accepted 22 August 2020
Available online 8 September 2020
1084-8045/© 2020 Elsevier Ltd. All rights reserved.

is used to carry out malware analysis, requires protocol models and message formats so that interaction with the attacking endpoint becomes possible (Wang et al., 2012). Vulnerability discovery tools also leverage the protocol behavioral models, such as finite state machines (FSMs), to produce illegitimate and unexpected patterns (Bossert et al., 2014). For example, in order to reveal potential vulnerabilities of a targeted application, the knowledge of a wrong sequence number, acknowledgment number, and invalid transitions are essential.

Most of the time it is relatively easy to obtain the network traffic traces for protocol reverse engineering analysis from an Internet Service Provider. However, in behavioral model reconstruction, current implementations tend to focus only on the capture of control flow aspects as a Finite State Machines (FSM), without the reference to those in data flow (Duchene et al., 2018). Moreover, the message format inference result, enforced by the message type identification, is not accurate in both a keyword analysis approach and distance-based methods adopted in the construction of the FSM.

To improve the FSM performance, a behavioral model termed the Extended Finite State Machines (EFSM) and consisting of both data flow and control flow information, is proposed in this paper. An EFSM, comprising memories in states and data guards annotated in conditional transitions, can represent more specific and correct behaviors with a significant reduction of states compared to those of the FSM (Lorenzoli et al., 2006). Extended Finite State Machine (EFSM) inference is a promising solution to the behavior exhibited by traditional FSMs: the exhibiting behavior of FSM depends on an internal state (Foster et al., 2018). In other words, with the help of data guards and memories, an EFSM makes the transition conditional, as data checking is required before transitions are allowed (Petrenko et al., 2004).

1.2. Contributions

In this paper, we present the pioneering work on methodologies and system implementation that employ network traces as the input to infer the EFSM of given protocols. The processing flow chart of the proposed ReFSM, as shown in Fig. 1, is composed of four main module: *Data Preprocessing, Message Type Identification, FSM Reconstruction, and Semantic Deduction.*

The Data Pre-processing module takes the raw packet data, reassemble it, and extracts network sessions. Abnormal messages removal and missing messages handling are part of the process. Based on the hybrid methodologies of Apriori and K-mean, the message type identification is responsible for clustering the messages from the data of the network sessions. The FSM reconstruction module infers the states and transitions of the finite state machine. It further minimizes the number of states by using k-tail merging methodology. The inter-message and intra-message dependencies are significant to the inference of the behavioral communication model. Therefore, the semantic deduction module is designed to detect the dependencies of these messages and finds the data guards and memories.

The main contributions of this work can be summarized as follows.

- ReFSM, a novel method to infer the EFSMs of network protocols, is proposed by considering not only the aspects of the control flow but also the data flow information from network traffic traces.
- A hybrid message type identification method consisting of keyword analysis and distance-based methods is proposed to overcome the isolating limitations for accuracy enhancement of inferred FSM.
- The methodologies of semantic deduction and the correlation detection technique are presented to infer inter-message and intra-message dependencies.
- This paper presents K-tail state merging to minimize the number of states and perform Daikon and Pearson product-moment for dependencies analysis.
- The use of *data guard* and *memory* is demonstrated with the accuracy-enhanced FSM for the proposed EFSM construction.

The structure of this paper is as follows. Section 2, gives a general background to the methodologies of protocol reverse engineering. The proposed methodologies and implementations concerning data pre-processing, message type identification, FSM construction, and semantic deduction are described in Section 3 along with the problems addressed in this paper. In particular, Section 3.5 presents a detailed discussions on the major steps in the semantic deduction module, which produces the EFSM. Section 4 discusses the setup of overall system testing and evaluation strategy. Detailed discussion of complexity comparisons are provided in Section 5. Section 6 presents the experiment results and discussions related to performance and complexity. Finally, in Section 7, we summarize the work presented and make suggestions regarding future work.

2. Background and related work

The ultimate goal of network-traces based protocol reverse engineering is to determine both message formats and behavioral models of communication protocols by trace analysis. Therefore, methodologies generally proposed can be categorized into message type identification, message format inference, semantic deduction, and behavior model construction (Kleber et al., 2019; Sija et al., 2018).

The ScriptGen tool (Leita et al., 2005) relies on the PI project (Beddoe, 2018), which uses bioinformatics techniques to infer message format. PI uses the Needleman and Wunsch (N&W) (Needleman and Wunsch, 1970) sequence alignment algorithm not only to calculate distance score for message clustering (message type identification) but also to identify common parts of messages in the same cluster for the format of messages exposure. This concept is further developed in SciptGen and extended to reconstruct deterministic finite state machines to generate a set of scripts for a honeypot, which can mimic a protocol to interact with attackers. Based on the same approach, Netjob (Bossert et al., 2014) modifies N&W and Unweighted Pairwise Groups with Arithmetic Averaging (UPGMA) hierarchical clustering algorithm (Sokal and Michener, 1958) with the help of contextual information to enhance the accuracy of message type identification. To this end, the authors developed a framework that allows actively collecting traces with context information.

The main idea proposed by Kleber et al. (2020) is to use vector distance on unequally-sized feature vectors by combining two popular methodologies of Hirschberg alignment and the DBSCAN cluster algorithm. However, Kleber et al. only focused on the message type identification, which is just one part of whole protocol-model reverse engineering. The system output is a set of message types of protocol. These authors are not concerned with the control and data flow of a protocol. Moreover, only stateless protocols are evaluated in the system. In another approach, ReverX (Antunes et al., 2011) and AutoReEngine (Luo and Yu, 2013) leverage keyword analysis to re-build a Finite State Machine. While ReverX constructs a graph by a generalization heuristic and then reduces it by conventional FSM minimization, AutoReEngine applies a popular data mining technique, *Apriori*, to find the most frequent string of bytes with keyword analysis for FSM construction. Without further minimization, the AutoReEngine model often contains a large number of states.

In contrast, Veritas (Wang et al., 2011) and PRISMA (Krueger et al., 2012) only focus on the construction of a behavioral model of protocols. The reconstructed behavioral model of Veritas is a probabilistic protocol state machine, which is a form of non-deterministic FSM with the probability of transitions. In message type identification, Veritas also uses PAM for clustering with Jaccard Index (Jaccard, 1912) as a distance score. Similarly, PRISMA itself defines a distance metric following an embedding concept in natural language processing. The messages are split into tokens by predefined delimiters (textual protocol) or fixed-size delimiters (binary protocol). It is then further mapped to vector space to directly calculate distance score applying a Euclidean metric.

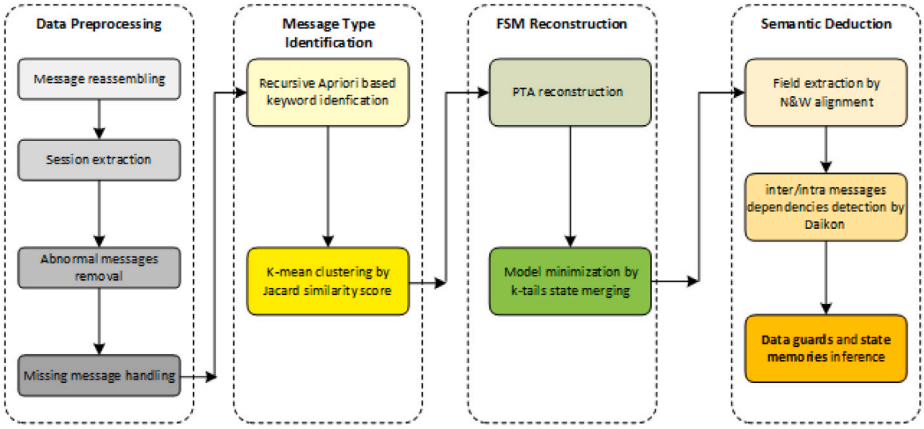

Fig. 1. The system architecture of the proposed ReFSM. The *Data Pre-processing* module is designed to clean, reassemble and extract sessions. The *Message Type Identification* is responsible for clustering the incoming messages. The *FSM Reconstruction* module is used to infer the states and transitions of the finite state machine. The main tasks of *Semantic Deduction* module are to find the data guards and memories.

The MINT (Walkinshaw et al., 2016) technique proposed by Walkinshaw et al. initiates a Prefix Tree Acceptor (PTA) from execution traces. The PTA is then minimized and generalized by the Evidence-Driven State Merging algorithm to obtain the control part of the EFSM. The state pair with the highest scores of the transaction in the outgoing paths of the same label is likely to be merged. To infer the data guards, the author proposed the use of a broad family of data classifier algorithms in the machine learning domain.

Lo et al. presents (Lo et al., 2012) an empirical setup that compares four model-inference techniques from software execution traces: K-Tail (Biermann and Feldman, 1972), kBehavior (Mariani et al., 2011), GK-Tail (Mariani et al., 2017) and KLFA (Mariani and Pastore, 2008). Similarly, the work of Krka et al. (2014) provides an empirical study that compares the quality of the inferred FSM model of four approaches: traces only, invariant only, invariant-enhanced-traces and trace-enhanced-invariants. The results highlight the benefit of combining the trace-based approaches and invariant-based approaches.

The methodology proposed by Goo et al. (2019) leverages another data mining technique called Continuous Sequential Pattern (CSP) to extract the probabilistic FSM of plain-text protocols. A hierarchical and recursive CSP is used in entire methods to infer field format, message types, and also the flow pattern that form the FSM. The CSP algorithm, modified from the Generalized Sequential Pattern (GSP) (Srikant and Agrawal, 1996), only finds the continuous sequences with the restricted orders of items. This constraint of elements in sequences helps to find the keyword (static field) of the message more effectively. However, the protocols (HTTP & DNS) evaluated are stateless, not stateful. The authors do not mention data part and only focus on the control part of the protocol. Moreover, the proposed method does not perform any state merging to reduce the number of states and transitions, and the size of the FSM may thus be insignificant. They do not process the data flow, only infer the FSM by using the control ones.

Kleber et al. (2019) Sija et al. (2018) and Narayan et al. (2015) provided an extensive survey on these significant subjects of protocol reverse engineering. In these papers, the authors compare the related works in this field and present the detail of analyzing the technologies used in each step: pre-processing, message type identification, message format inference, and protocol model reconstruction.

Table 1 presents the summary of all these selected works in protocol reverse engineering from network traffic traces.

2.1. Extended finite state machine

A good behavioral models of protocols should capture not only the order of exchanging messages (control flow aspect) but also the constraints on the data being transmitted (data flow aspect). One of the most widely-used models is the Extended Finite State Machines, which enhances Finite State Machines by embedding memory in the states and data guards in the transitions (Lorenzoli et al., 2006; Petrenko et al., 2004; Walkinshaw et al., 2016). An Extended Finite State Machine is typically defined as a 6-tuple EFSM=$(S, I, O, \sigma, \delta, M)$, where S is the set of finite states, I is the set of inputs, O is the set of outputs. M is the set of memories equipped to states. A set of Boolean expression functions $\delta : I \times M \rightarrow true, false$, defined in Eq. (3), is termed as data guards, and represents the data constraint validation before state transitions. The symbol $\sigma : S \times I \times M \rightarrow S \times O$ is the set of transitions regulating that at state $s \in S$ and corresponding $m \in M$, if the machines accept the input as $i \in I$, the machines enter to state $s' \in S$ and produce output $o \in O$.

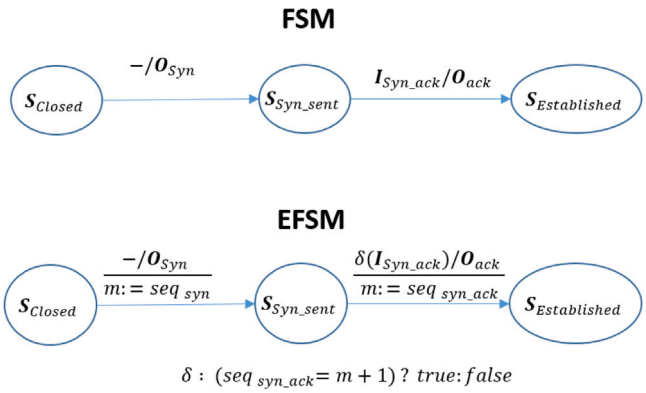

Fig. 2. Example of the partial EFSM and FSM state diagrams for the TCP protocol. The EFSM is capable of checking TCP sequence numbers in the transactions. In EFSM, sequence number of previous message is stored in memory of state of syn_sent, the data guard of this state will compare the sequence number of incoming messages with memory before perform the transition.

Taking TCP handshake as an example, an EFSM uses a data guard to check the acknowledged number received and sent sequence numbers in memory. Fig. 2 shows an example of EFSM and FSM as a part of TCP operation. After sending *syn* messages, the client machine enters the *syn_sent* state, writes the sending sequence number to memory, and waits for a response. When the *syn_ack* message is received, the client needs to check whether the received sequence number is equal to the value plus one in memory so that transiting to the established state can be allowed.

Table 1
The comparisons of related works in protocol reverse engineering from network traffic traces. The detailed discussions of complexity comparisons are provided in Section 5. *LS*: Limited support, *S*: support.

Methodology	Message type		Behavioral models reconstruction		Message Format Inference	
	Identification*		Models	Algorithms	Algorithms	Semantic Deduction
Proposed	Hybrid	Apriori + K-mean	Extended FSM (ReFSM)	K-Tail	Sequence Alignment	S
AutoReEngine (Luo and Yu, 2013)	Frequent String Extraction	Apriori (k-length constraint)	FSM	Sequence Labeling Apriori	Keyword Group Extraction	N/A
CSP (Goo et al., 2019)	Hierarchical CSP	Improved Apriori	FSM	Recursive CSP	FieldHunter's semantics inference	*S*
Discover (Cui et al., 2007)	Type based	Tokenization Recursive Clustering	N/A	N/A	Type-based Sequence Alignment	*LS*
NetJob (Bossert et al., 2014)	Distance based	K-mean (N&W alignment score)	FSM	N/A	Sequence Alignment	*LS*
PRISMA (Krueger et al., 2012)	Distance based	K-medoids clustering+Jaccard Index	FSM	Generalization	N/A	N/A
ReverX (Antunes et al., 2011)	Keyword analysis	PTA + Frequency Labels	FSM	Generalization and Minimization	Graph generalization	N/A
ScriptGen (Leita et al., 2005)	Distance based	Modified N&W alignment	FSM	Generalization and Minimization	Sequence Alignment	N/A
TPRE (Trifil et al., 2009)	Statistical Keyword analysis	Variance of the Distribution	FSM	State Splitting	N/A	N/A
Veritas (Wang et al., 2011)	Distance based	Kolmogorov–Smirnov Statistical testing	Probabilistic FSM	N/A	N/A	N/A

2.2. Daikon and K-Tail algorithm

EFSM inference of communication protocols from only network traces has not yet been addressed, but several pioneering techniques have been proposed to infer EFSM model of software from software execution. The first method to attempt this issue is GK-Tail of Lorenzoli et al. (2006)(Lorenzoli et al., 2008). In this work, the authors used the K-Tail algorithm to infer the control part (FSM). The Daikon invariant detection system is adopted to derive the data guards as the extended part of the FSM (Mariani et al., 2017).

The K-Tail algorithm is a model reduction algorithm which helps to simplify the complicated models by merging their equivalent states. The basic idea of the K-Tail algorithm (Biermann and Feldman, 1972) is that two states are equivalent if they share the same behavior in the next k transitions. An example of k=2 is illustrated in Fig. 3. The state S_2^0 and S_2^1 are equivalent because they share the same 2-tails $\{(i_3/o_3),(i_4/o_4)\}$, which is similar to the pair of (S_3^0, S_3^u) because their 2-tails is $\{(i_4/o_4),(i_5/o_5)\}$. The result of this algorithm is the reduced FSM in Fig. 4. This heuristic is effective for reducing non-deterministic FSMs where the standard deterministic FSM minimization algorithm fails. Software reverse engineering methods typically used it to obtain minimized models.

Daikon (Ernst et al., 2001) is an implementation of dynamic invariant detection. It is a template-based, machine learning technique that can be applied to arbitrary data. It can take the raw execution traces or the values of variables as input and finds the best matching properties (rules) for all observed values of variables. To illustrate the outputs of Daikon, we denote x, y as variables and a, b as constant. Taking the input of all observed values of x, y, Daikon reports a simpler invariant (a constraint) with related constants a, b. For a single variable, it discovers a constraint that holds over its values. Daikon can produce simpler invariant such as being a constant ($x = a$), in enumerative sets of values ($x \in a, b$), or in a range that is restricted by values of minimum and maximum ($a \leq x \leq b$). For multiple variables, it finds a correlation between the values of variables. It can also determine constraints based only on the dataset containing observed values of the variable so that it can be used to infer the data guard of EFSMs.

2.3. Message type identification

Message type identification, a core component of protocol reverse engineering, has significant effects on the accuracy of final results. It also creates an abstraction of messages similar to group messages that are semantically related. Each group contains many messages which share the same structure pattern and similar purpose. Taking the FTP protocol as an example, the "USER" message type always contains the expression "USER <email>" and is used to authorize the username of systems. Current studies based on network traces often rely on *keyword-based* or *distance-based* clustering algorithms to identify protocol message types.

Keyword-based methods mostly leverage the frequently-occurring strings, also named as keywords (e.g., "USER", "PASS") to distinguish message types. They use statistical tests such as the Kolmogorov–Smirnov test, Statistical t-test, Apriori (Agrawal and Srikant, 1994), and Distribution of Variances (Trifil et al., 2009) to extract distinctive keywords.

ReverX (Antunes et al., 2011) adopted the frequency labels, a keyword-based analysis methodology to construct the Prefix Tree Acceptor (PTA). It assumed the message fields are usually split by predefined delimiters. The authors of TPRE (Trifil et al., 2009) assumed that there is a command/control byte (bit) that is located in the fixed position to represent the message type. The major cost of the proposed methodology is the computation of the Variance of the Distribution of the Variances (VDV) for every position in messages.

NetJob (Bossert et al., 2014) used the K-mean algorithm with the N&W alignment score as distance metric. In ScriptGen (Leita et al., 2005), the authors used the same methods of Netjob with modified N&W alignment. PRISMA (Krueger et al., 2012) adopted the k-medoids clustering algorithm with Jaccard index as distance metric.

As a result of applying the frequency of a string, this kind of method often ignores unique message types that rarely occur (e.g., "PWD", "QUIT"). Another drawback is keyword misperception: There are some terms used in hot topics that appear frequently and are easily confused with actual protocol keywords. These limitations lead to the inaccurate identification of message types, as incorrect messages are grouped into different clusters.

$$Ses_1 = \{(i_1, o_1), (i_2, o_2), (i_3, o_3), (i_4, o_4)(i_5, o_5)\}$$
$$Ses_2 = \{(i_1, o_1), (i_3, o_3), (i_3, o_3), (i_4, o_4), (i_6, o_6)\}$$
$$Ses_3 = \{(i_7, o_7), (i_8, o_8), (i_9, o_9), (i_4, o_4), (i_5, o_5), (i_{10}, o_{10})\}$$

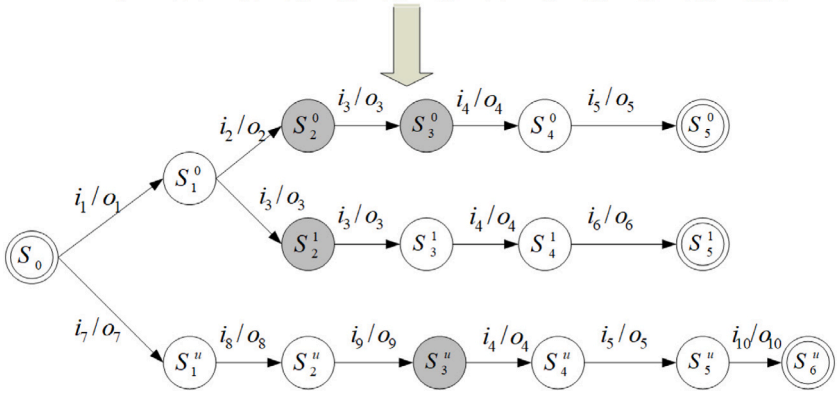

Fig. 3. The initial FSM obtained from sessions of message sequences. For every input sequence, the process starts from the root and travels down along the tree. A new path is created if there are no existing paths. The state S_2^0 and S_2^1 can be merged because of the shared 2-tails of i_3/o_3 and i_4/o_4. The pair of states S_3^0 and S_3^u can also be merged because of the shared 2-tails of i_5/o_5 and i_4/o_4.

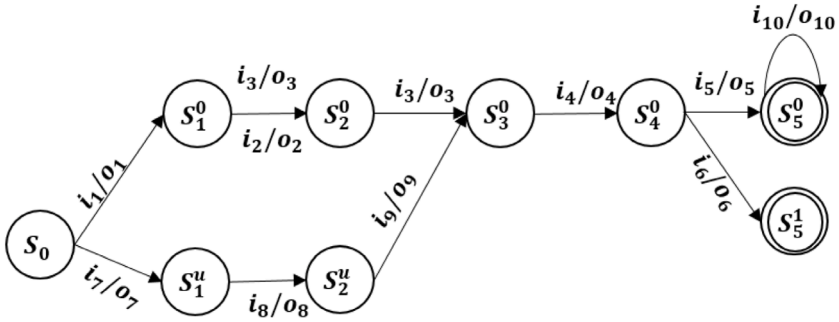

Fig. 4. The new FSM after merging by K-Tail mechanism with K=2 from the FSM shown in Fig. 3. The state S_2^0 and S_2^1 is merged as S_2^0. The pair of states S_3^0 and S_3^u is also merged as S_3^0.

Distance-based methods apply an unsupervised machine learning algorithm with a similarity metric of messages for message clustering. This approach merges pairs of clusters by their similarity. Among many machine learning techniques, unweighted pairwise groups with arithmetic averaging (UPGMA) and Partitioning around Medoid (PAM) are the most popular. Several metrics have been proposed, such as the Jaccard index (Jaccard, 1912), Needleman and Wunsch (N&W) (Needleman and Wunsch, 1970) or the Longest Common String (LCS). The disadvantage of this approach is that a knowledge of the number of clusters is required in advance. Sometimes, it is hard to define. Otherwise, these methods are easy to apply under specified or over specified clustering in practice.

2.4. Inter and intra message dependencies

The term "semantic deduction" refers to a process that infers inter- and intra-message dependencies that exhibit the semantic meaning of fields in messages. The term inter-message dependency represents the relationship of a field that regulates the property of another field in different messages, such as cookies (HTTP) and sequence number (TCP). Intra-message dependency is a correlation between fields within one message, for example, consistency fields as check-sum or direction field as length and offset.

Although inter and intra-message dependencies play an essential role in building the right messages and interactions, only a few existing works can support such inferences. As a result of a massive amount of possible dependencies, all approaches eventually leverage heuristics for searching with the support of human interpretation. Finding the most matches in a predefined set of rules is offered by PRISMA (Krueger et al., 2012) and ScriptGen (Leita et al., 2005) for the semantic deduction. However, they require a few manual interpretation step to establish a set of rules and can only cover a little semantics.

3. ReFSM System Architecture

The ReFSM processing model, consists of four major modules: *data pre-processing*, *message type identification*, *FSM construction* and *semantic deduction*, is illustrated in Fig. 1. First, the traffic traces of a particular protocol have to be pre-processed in steps: message reassembly, session extraction, and cleaning. The message type identification module uses the Apriori keyword analysis to extract protocol keywords and to determine the number of the clusters before clustering messages into groups by k-means algorithm. Each group is considered as a different message type. The results are then fed to the FSM construction module to infer the FSM by using initialization and K-Tail merging algorithm. After the correct FSM has been determined, the semantic deduction module extracts sub-datasets containing values of message fields in observed messages. Further analysis is performed to search the correlation of fields in the messages. The result, thus deduced, is used to form the data guards on each transition in the EFSM.

3.1. Problem statement

Given a set of network traces Tr containing sessions of target protocol, the objective of our work is to reconstruct an accurate behavioral model for protocol in the form of EFSM with six-tuple (S, I, O, σ, M and Δ) information. The greater correctness and coverage scores reflect the quality of the model.

Table 2 lists the notations used in this work. A packet, denoted as Pk is the primary element of communication between two entities. The session $Ses = \{Pk_i | i = \overline{1, N}\}$ is the ordered sequence of packets; it is defined by its five-tuple source address, source port, destination address, destination port and timestamp. The input in our system is the network traces $Tr = \{Ses_j | j = \overline{1, K}\}$ which consists of set of sessions. We assume that each session only contains information of the target protocol. The set of states, inputs, outputs, transitions, memory and data guard of EFSM are denoted by I, O, σ, M and Δ. The $EFSM^{inf}$ represents the inferred model reconstructed from network trace Tr. The $EFSM^{true}$ represents the actual model extracted from the exact specification of the target protocol.

3.2. Data pre-processing module

This module takes traffic traces from TCP dump files as input. At the beginning of this process, messages are parsed to extract sessions based on the 5-tuple packet header fields: source address, source port, destination address, destination port and type of transport layer protocol. The fragmented messages are then assembled, and the duplicates and re-transmissions are removed. The remaining messages continue to be parsed by ignoring unrelated messages; only the payloads containing the information of the target application protocol are kept for further analysis.

3.3. Message type identification module

Hybrid keyword-analysis and distance-based approaches are used to increase the quality of message type identification.

Based on the modification of AutoReEngine (Luo and Yu, 2013), the Apriori keyword analysis is used to identify the keywords. The algorithm interactively finds the high frequency and close sequences of bytes (or string) with stable position variance by the Apriori method. In each iteration, only closed sequences with a frequency higher than a pre-defined threshold are defined as keywords.

After the keywords extraction process, the keyword series observed in the dataset can be used as the unique format of message types. Each keyword series is a group. The extracted keyword series resolve the limitation of having the number of clusters in advance of k-means. Also, the number of clusters is also determined before the k-means algorithm. The distance metric is based on the Jaccard index (Jaccard, 1912) defined as $1 - \frac{|a \bigcap b|}{|a \bigcup b|}$ where a, b are the character array of the messages. The issues of threshold and keyword misperception are resolved by the iterative k-means clustering based on the distance metric. The k-means clustering algorithm helps to calibrate the keyword extraction in the first step. The undecided message sets are kept to overcome the issue of a missing keyword so that the extraction process can be repeated. Because the size of the dataset is reduced, the keywords which have a low occurrence in the original dataset can be revealed. For instance, in Fig. 5, the keywords "QUIT" and "RNFR" are recovered in the second iteration. The procedure ends when the undecided set is empty or acceptable.

3.4. FSM construction module

The goal of this module is to infer a conventional FSM model with 4-tuple: (S, I, O, σ) before the construction of the EFSM model. The proposed methodology consists of two parts: the initialization and state merging.

As network traces are collected from real-world traffic, some sessions are incomplete, and messages have ordering issues due to packet loss and network delay. Thus, data cleaning techniques such as sequence reorder and missing value inference needs to be applied to achieve a better result for the FSM construction.

A message requested by a client and a response from a server can be represented as (i_k, o_k), where $i_k, o_k \in T$ and T is the set of message types. The session can be represented as a sequence of messages $Ses = (i_1, o_1), (i_2, o_2) \ldots (i_m, o_m)$. In the first step of processing, a dataset is fed to the initialization module to build a Prefix Tree Acceptor, which accepts all session sequences. For every session sequence, the process starts from the root and travels down along the tree. As shown in Fig. 3, a new path is created if there are no existing paths.

Subsequently, the initial FSM is interactively refined by merging equivalent states based on the K-Tail mechanism. The intuition is that the protocol state machine always exposes the same behaviors in the same state. In other words, the machine produces the same outputs if the same inputs are submitted at a particular state. Because of the deterministic characteristic of protocol state machines, the next subsequent states in K-Tail of two equivalent states can be merged as illustrated in Fig. 4.

The state merging procedure is defined in Algorithm 1. For each state $s \in S$, we define k-tail(s) as the set of next transitions. Pairs of states are all compared iteratively to initialize a list of sets consisting of equivalent states. Based on the list available, two sets are merged if they share at least one state in common until no two sets can be merged. At the end of this step, we obtain a list of sets consisting of equivalent states, and no two sets consist of the same state. For every set, a new state is created as the representative for all states. Then, for every original transition, a new transition is added between two new states. These represent the *start* and *end* states of the initial FSM.

Algorithm 1 K-Tail merging algorithm

Merge$(a, b) = a \cup b$
Input :S set of states, T set of transitions
Output: S', T'
Initialize Q=ϕ
Foreach pair $s_i, s_j \in S$ **do**
  if (k-tails(s_i) == k-tails(s_j)) **then** $Q \leftarrow Q \bigcup (s_i, s_j)$
While STOP_FLAG **do**
  STOP_FLAG ← FALSE
  Foreach $q_i, q_j \in Q$ **do**
    if $q_i \bigcap q_j \neq \phi$ **then**
      STOP_FLAG ← TRUE
      $Q \leftarrow Q - q_i$
      $Q \leftarrow Q - q_j$
      $Q \leftarrow Q \bigcup Merge(q_i, q_j)$
Endwhile
Foreach $q_i \in Q$ **do**
  $s_i' \leftarrow newState()$
  $S' \leftarrow S' \bigcup s_i'$
Foreach $t_j \in T$ **do**
  $t_j' \leftarrow newTransition()$
  Foreach $q_i \in Q$ **do**
    if $t_j.startState \in q_i$ **then** $t_j'.startState \leftarrow s_i'$
    if $t_j.endState \in q_i$ **then** $t_j'.endState \leftarrow s_i'$
    $T' \leftarrow T' \bigcup t_j'$

Table 2
Table of notations.

Category	Notation	Description
Entity	Pk_i	The i_{th} packet
	$Ses = \{Pk_i \mid i = \overline{1,N}\}$	The session is sequence of packets.
	$Tr = \{Ses_j \mid j = \overline{1,K}\}$	Traces is set of sequences
Process	$S = \{S_i\}$	The finite set of states
	$I = \{I_l\}$	The finite set of inputs
	$O = \{0_k\}$	The finite set of outputs
	$M = \{M_{s_i}\}$	The finite set of memories on states
	$\Delta = \{\Pi \mid \Pi_l : I \times M \rightarrow \{0,1\}\}$	The data guards
	σ	The set of transitions
	$EFSM^{inf}$	The six-tuple of inferred EFSM
Evaluation	SI^{true}	The sequences of true model 's input
	SI^{inf}	The sequences of inferred model 's input
	$EFSM^{true}$	The six-tuple of true EFSM
	$EFSM(SI)$	The sequences of output generated by submission SI to $EFSM$
	$Cov = \frac{EFSM^{inf}(SI^{true})}{\lvert SI^{true}\rvert}$	The coverage: How many sequences of true input are accepted by inferred model. Coverage ⇔ Probability $(EFSM^{true} \subset EFSM^{inf})$
	$Cor = \frac{EFSM^{true}(SI^{inf})}{\lvert SI^{inf}\rvert}$	The correctness: How many sequences of inferred input are accepted by true model. Coverage ⇔ Probability $(EFSM^{inf} \subset EFSM^{true})$
	m	The average number of messages in a single cluster.
	n	The number of messages.
	l	The max length (bytes) of a single message.
	d	The total number of possible fields.
	t	The number of states in the Prefix Tree Acceptor (PTA).

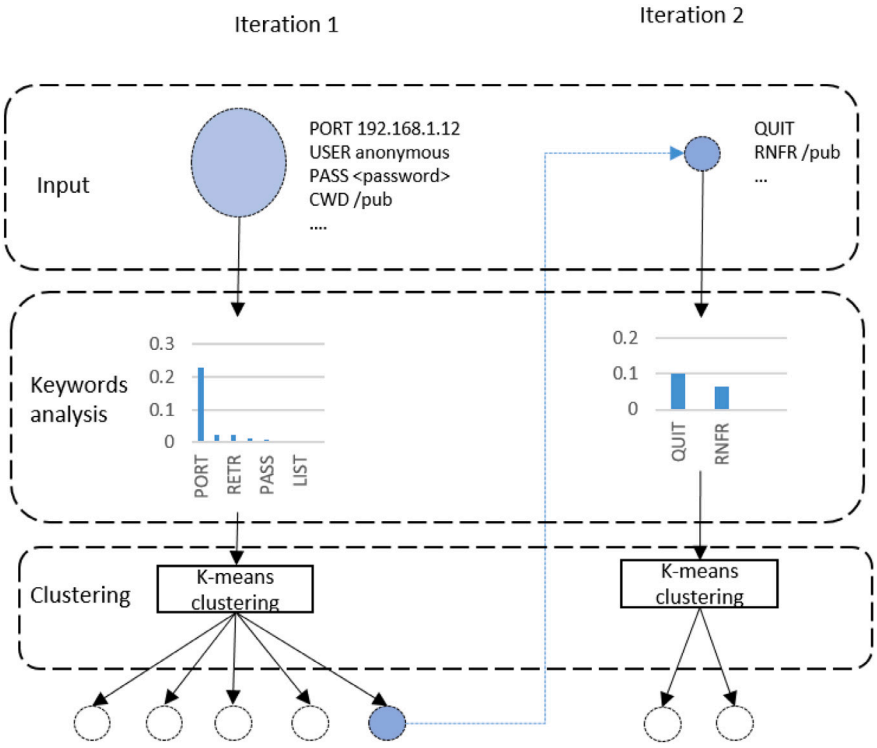

Fig. 5. A typical example of the processing flow for message types identification with two iterations on the FTP dataset. After the first iteration, with high occurrence frequency, the keyword of PORT, RETR, PASS and LIST are discovered. The rest of the keywords will be processed in the second iteration.

3.5. Semantic deduction module

After the 4-tuple (S, I, O, σ) finite state machine is constructed, we start to generate the data guards and memories (δ, M) for each transition to form the 6-tuple $(S, I, O, \sigma, \delta, M)$ extended finite state machine. In the beginning, the module identifies the type of message and decompose the messages into fields. Then it validates the values of each field to decide whether further actions should be performed.

3.5.1. Fields extraction

At this stage, the format of each message type is inferred to obtain fields and the sub-dataset. For every cluster of different message types, the multiple Needleman and Wunsch sequence alignment algorithm (Bossert et al., 2014) is used to extract the format. As a result of the high computational complexity of this algorithm, a progressive alignment (Durbin et al., 1998), based on a pre-build guide tree to decide the order, is used for the alignment process. The results are composed of a consensus string, dynamic fields, and static fields. The static fields are similar to those protocol keyword series extracted

before. A sub-dataset for each dynamic field is built by obtaining all observed values in the traces and grouping by the response message type.

Take the FTP protocol for example, given a message of "*PORT 65,240,180,205,56,56*", the FTP server identifies that the request command is *"PORT"* and the data portion is "*65,240,180,205,56,56*". The first four numbers of the data portion are identified as the encoding of IP address and the remaining are the port number. The FTP server further checks that the port number is within the range of 0 and 255. Based on this, a field extraction is adopted to build the sub-dataset containing all values of fields before ReFSM uses Daikon to infer the constraints hold the overall values of a certain field. An illustrated example of PORT message alignment is given in Fig. 6. The left-hand side presents the messages in traces and the N&W alignment progress. The right-hand side shows the sub-dataset of each field in the traffic traces corresponding to the response code of "200". The sub-datasets of fields are then mined to deduce the protocol semantics.

3.5.2. Deriving data guards

Assuming that the message $i = [f_1, f_2, \dots, f_t]$ consists of fields f_t. The data predicate of a field is defined as

$$Pr_u(v) = \begin{cases} 1 & if\ v \in Daikon\,(f_u) \\ 0 \end{cases}. \tag{1}$$

As mentioned above, the data guard δ act as predicate on the input message. Then the data guard function $\delta(i, m)$ can be defined as

$$\delta(i, m) = \delta(f_1, f_2, \dots, f_k, m) \tag{2}$$

$$\approx \bigwedge_1^t Pr_u(f_u) \bigwedge_1^t Itr_{uv}(f_u, f_v) \bigwedge_1^t Ite_{um}(f_u, m). \tag{3}$$

The parameters are listed as follows:

- $Pr_u(f_u) \rightarrow 0, 1$ is a data predicate of field $f_u$for data validation. As illustrated in Eq. (1), this function checks whether the value of field f_u is valid or not.
- $Itr_{uv}(f_u, f_v) \rightarrow 0, 1$ is a constraint between values of two fields f_u, f_v, which is also known as the intra-message dependency.
- $Ite_{um}(f_u, m) \rightarrow 0, 1$ is a constraint between value of field f_u and the memories m stored in previous message. It is the inter-messages dependency.

For example, the IP address and port number format in the argument of FTP's PORT command are regular expressions of the first type. The *check-sum* and *direction* fields, such as length and offset, are the intra-message dependencies. The cookies in the HTTP protocol are inter-messages dependencies.

3.5.3. Constraints on fields

ReFSM relies on the Daikon (Ernst et al., 2001) algorithm to generate the data guard of each field. Daikon deduces a broad set of values in sub-dataset into a simpler invariant for ReFSM to use as a data guard.

Principally, $Daikon()$ takes the raw traces or the values of variables f_u as input and finds the best matching properties (rules) for all observed values of variables. As shown in Fig. 7, Daikon worked on the sub-dataset of field f_1 of PORT command data and found that the value of f_1 must be in the range of 0 and 255. Similarly, the arguments of the TYPE command are in the enumerable set of {"A," "I"}. If this constraint is not verified, the finite state machine may replies code of 500 instead of code of 200.

3.5.4. Inter/intra message dependency

The proposed methodology continues to identify relations between fields in one or two messages for the formulation of a data guards function. The core process leverages the Pearson coefficient representing the strength of dependency on all pairs of attributes in the fields. Potential candidates are then selected by using Daikon to infer the relationships.

At first, the dataset of attributes is prepared. The process consists of the field's value, length, and offset in the protocol message to capture a different kind of relationship. Note that other fields such as IP address and port number are also taken into account. All observed pairs of field attributes are computed interactively over sessions in the network traffic traces.

Once this has been carried out, the Pearson product-moment correlation coefficient $\rho(X, Y)$ is computed based on each observed pair of field attributes (X, Y). The absolute value of the correlation coefficient $|\rho(X, Y)|$ indicates the strength of the relationship. If the value is close to one, there is generally a linear correlation between X and Y. If the value is close to zero , they are mostly independent. For instance, the correlation coefficient value is close to one on the two attributes of Content-Length (HTTP header) and Length (IP header) because they are linear-dependent. Finally, these dependencies are simplified by applying the Daikon algorithm so that a linear relationship of $(Y = aX + b)$ can be derived. The dependencies between fields in the same message are classified as the Itr function. For the dependencies between fields in two messages are classified as the Ite function.

3.5.5. Transform FSM to EFSM

As mentioned above, we need to infer the 2-tuple data guard and memory (δ, M) on each transition of the FSM. The data guard function δ, as illustrated in Eq. (3), can be inferred by merging the Pr, Itr and Ite functions. The values of fields are updated and kept in the memory of each state for future interactions. A field's values in previous messages can be assigned and kept in the memories of states. However, the size of the memory required is enormous, mainly because of the sequences of the messages that need to be processed. A simple heuristic on the pair of inter-message dependencies can be adopted to further reduce the size of the memory. Initially the field values of every message are kept in the memory temporarily, and can be eliminated if values are not used in the following messages. Once one of the fields in the inter-message dependencies pairs appear in the future, the corresponding value in the memory is kept. The memory in the states completes the 6-tuple of EFSM.

4. Evaluation

This section details the system setup and experimental results of the proposed ReFSM. Its purpose is to evaluate the quality of the inferred EFSM on a given protocol from network traces.

For this evaluation, we use the *precise* as correctness score and *recall* as coverage score. The sequence generated by the exact model SI^{true} and the inferred model SI^{inf} is handled by the model of $EFSM^{inf}$ and $EFSM^{true}$, respectively, to obtain output. The coverage score represents how the inferred model accepts many sequences of valid input. The correctness is used to represent how many sequences of inferred input are accepted by the actual model. The acceptance means the final state reachability of each input.

In order to measure the effectiveness of the proposed ReFSM, metrics of the pair-wise Precision and Recall (Hatzivassiloglou and McKeown, 1993) values are used to evaluate the effectiveness of clustering algorithms for message type identification. For the quality of inferred EFSM, the correctness, coverage score, and behavioral accuracy are adopted for comparison to the conventional FSM. AutoReEngine (Luo and Yu, 2013), relying on the Apriori algorithm (Agrawal and Srikant, 1994) with a pre-defined threshold and the variances of position to filter out keywords is selected for the comparative study. The experimental setup, including datasets and metrics, is presented first. Then, the evaluation results and the analysis are described in detail.

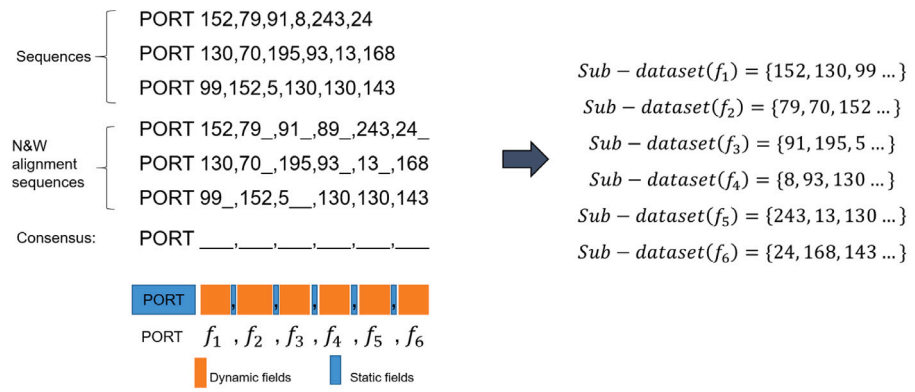

Fig. 6. An example of Needleman and Wunsch (N&W) alignment sequences for field extraction. After the alignment procedure, the format of the message type PORT becomes *port* $f_1, f_2, f_3, f_4, f_5, f_6$, where f_i represents a variable. Those values of each field are used to build dataset for each field..

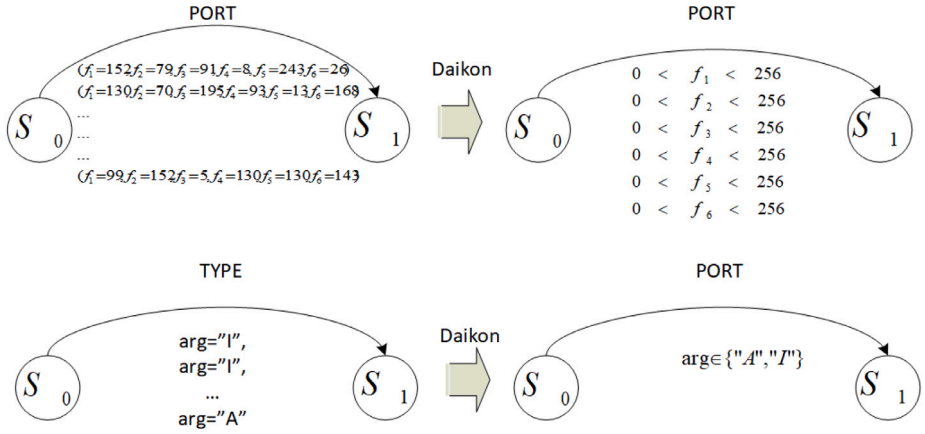

Fig. 7. An example presents the data predicates derived by Daikon. Daikon deduces the broad set of values on the left-hand side to a simpler invariant on the right side. In this example, Daikon infers the valid value of f_i in the range of (0,256), and the argument of TYPE message to be {'A','I'}.

4.1. Experimental setup

The prototype of ReFSM is implemented in Java language of approximately 6600 lines of code based on the libraries of jNetPcap, Libpcap and Daikon (Ernst et al., 2001). The system is capable of accepting network capture files in tcpdump format as input. The Daikon inference engine is adopted to derive constraints as data guards on the transition between states in the EFSM.

Four protocols of FTP, SMTP, BitTorrent, and PPLive are selected to evaluate the methodology proposed. The datasets of network traces are collected from publicly available and self-capture sources. The real-world FTP traces are used from Lawrence Berkeley National Laboratory (Pang and Paxson, 2003). The anonymized network traces consist of more than 210,000 FTP messages of 1,800 sessions. However, these traces still contain malformed messages, and extra steps to eliminate illegal messages are needed. The FTP traces are merged with networking traces captured from the NCTU university networks. The SMTP and BitTorrent datasets also are captured from the university network. Unveiling specifications of proprietary and closed protocols are one of the motivations of this work, so that PPLive, one of the most popular P2P video streaming application, is selected. Table 3 summarizes our datasets used in the evaluation.

Table 3
Summary of datasets used in the system performance evaluation.

#	Protocol	Source	Number of Messages	Number of Sessions
1	FTP	LBNL/Generated	217,281/5,235	1,841/563
2	SMTP	Generated	4,862	426
3	BitTorrent	Generated	3,289	260
4	PPLive	Generated	8,092	281

4.2. Pair-wise precision and recall

Pair-wise Precision and Recall is one of the most frequently-used scores to evaluate the effectiveness of clustering. It is used to measure the quality of the proposed message type identification method. We denote E as the sets of expected clusters, and R as the set of actual clusters as a result. We calculate the true positives tp by counting the number of shared pairs in E and R. The false positives fp represents the number of a pair in R but not in E. The false negatives fn represents the number of a pair in E but not in R. Then, the precision and recall are defined as $recall = \frac{tp}{fn+tp}$ and $precision = \frac{tp}{fp+tp}$. The expected clustering results are manually derived from the protocol specification. Taking FTP as examples, 33 commands in RFC 959 (Postel and Reynolds, 1985) can be extracted as 33 clusters of FTP messages of client sides. Similarly, there are 11 messages clusters in BitTorrent version 1.0 and 9 clusters in SMTP.

4.3. Coverage and correctness scores

In order to compare the performance of two FSMs, previous work (Wang et al., 2011; Trifil et al., 2009; Krueger et al., 2012; Antunes et al., 2011) relied on the correctness and coverage (also known as Precision and Recall, respectively) to compare the inferred FSM and the one deriving manually from protocol specification.

Coverage is how much of the specification has been inferred by the model. It is calculated by the percentage of actual sessions (generated

by the true model) accepted by the inferred model. Correctness is the percentage of inferred sessions (accepted by the inferred model) are genuinely accepted by the true model. The acceptance is defined as the last state's reachability. In this work, these scores are used to demonstrate the proposed FSMs part of the inferred EFSM is at least equivalent or better than those in other works on FSM inference.

4.4. Behavioral accuracy

The techniques of correctness and coverage are, however, inadequate to assess the effectiveness of the EFSM, because the data guards are not covered and the outputs are not taken into consideration.

Produced at states that exhibit the protocol's response when receiving the incoming request, the outputs are the only information that can be observed to verify the internal states of machines. These are required for the application of behavior-based models, such as conformance test-case generation and fuzzing tools (Dahbura et al., 1990). An illustrated example with the FTP protocol is given in Fig. 8. The same input sequences are applied to both models of FTP protocol and obverse the outputs sequences. The generated output code values are {331,230,200,221} while the expected output code values are {331,230,500,221}. The differences are the return codes of 500 and 200, where the port argument of the PORT command is invalid. The evaluation of previous works failed in this case because both models can still reach final states.

It follows then that, the metric inspired by the protocol conformance test-case generation method and software specification mining theme (Dallmeier et al., 2012) are adopted here. The set of test cases, consisting of the input sequence and corresponding expected outputs, is prepared in advance. These inputs are then submitted to the inferred model to obtain the outputs and compare these to the expected outputs. As described, the index of *behavioral accuracy*, defined as $BA = \frac{Number\ of\ accepted\ outputs}{Total\ outputs}$, is an important quality indicator for both syntax and semantics of an inferred model of the protocol.

4.5. Test methodology

First, the actual models of protocols are needed to compare with the inferred models. For known protocols, we manually derived the accurate models from standard specifications. The model of FTP was extracted from RFC 959. The models of BitTorrent and SMTP were derived from specifications. We adopted the model in the work of Veritas (Wang et al., 2011) for the test because no PPLive standard specification is available. To obtain an accurate result and eliminate the test set dependencies, the k-folds cross-validation (Kohavi, 1995), one of the most popular evaluation method in machine learning, was applied. The set of traces was randomly split into k partitions. Over k iterations, one partition was kept to generate test cases and the remaining partitions were used to infer the models. The final score is the average of scores of k iterations.

To calculate the coverage score, we needed a set of positive sessions, *i.e.*, a sequence of accepted inputs by the actual model. For example, a session that consists five FTP messages of "USER anonymous," "PASS <password>," "PORT 140,113,207,14,88,90," "RETR /pub/info2017" and "QUIT" (denoted as sequence of {USER, PASS, PORT, RETR, QUIT}) was accepted as the model that reached the final state. These sessions were submitted to the inferred models to calculate the number of sessions to be accepted.

To compute the correctness score, both the positive sessions and the negative ones rejected by the actual model were necessary. Thus we had to generate a negative session beforehand, one that was still accepted by our model but could potentially be rejected by the actual model. For this, we randomly selected and mutated positive sessions similar to the methods presented in work by Antunes et al. (2011). Principally, the mutation of accepted sessions such as swapping messages, deleting and adding more messages is repeatedly performed to generate rejected sessions. For instance, the session {USER, PASS, PORT, RETR, QUIT} is accepted but the sessions {PASS, USER, PORT, RETR, QUIT} and {USER, PASS, PORT, QUIT, RETR} were rejected. Finally, to compute the correctness score, the test set consisting of both accepted and rejected sessions were submitted to inferred models. The ratio of accepted to rejected sessions affect the correctness score. In this work, the accepted and rejected session ratios of 4:1, 9:1 and 2:1 were used. Due to limited space, the results shown in this paper were based on the ratio of 4:1.

5. Cost analysis and comparison

Most of the protocol reverse engineering tools are developed toward a specific design targets such as the inference of message format, reconstruction of the behavior model, or both of them for different use cases (Kleber et al., 2019). Table 1 lists the comparisons of selected works in protocol reverse engineering from network traffic traces. In particular, the core algorithms utilized for the message type identification are listed since it is the typical processing step commonly found in these works of literature.

The proposed ReFSM architecture, as presented in Fig. 1, realizes four of the primary processing steps targeting the use case of network test case generation. In the beginning, the system conducts the packet pre-processing to obtain the protocol messages. The cost is proportional to the number of messages n.

The ReFSM system uses two techniques: Apriori algorithm and K-means for the message type identification. The time complexity of the K-means clustering is $O(n^2)$. For the Apriori algorithm, the complexity is $O(\sum_2^l k(k-2)|C_k|)$, where $|C_k|$ denotes the number of items in the set containing sub-sequence of k-byte keywords (Tan et al., 2006) and l is the max-length of a message. Fortunately, armed with the pruning method of a specific threshold, the message type identification can be processed efficiently. The CSP is derived from the Generalized Sequential Pattern (GSP) (Srikant and Agrawal, 1996), an optimized Apriori algorithm, so that the keywords found by Apriori cover all keywords found by CSP with less computational complexity. In general, the empirical evaluation using real-world data sets indicates that the processing time of GSP is faster than that of the Apriori algorithm (Agrawal and Srikant, 1994). ReverX (Antunes et al., 2011) adopted the frequency labels, a keyword-based analysis methodology to construct the Prefix Tree Acceptor (PTA) for message type identification. NetJob (Bossert et al., 2014) used the K-mean algorithm with the N&W alignment score as distance metric. The cost of calculating the distance metric is $O(l^2)$, where l is the maximum length of a message. In ScriptGen (Leita et al., 2005), the authors used the same methods of Netjob with modified N&W alignment. Overall, the complexity is still the same. PRISMA (Krueger et al., 2012) took up the k-medoids clustering algorithm with Jaccard index as distance metric. The cost of Jaccard index is $O(l^2)$.

The finite state machine construction process consists of two significant steps: Prefix Tree Acceptor (PTA) generation and state merging. The system has to process every message to generate the PTA and therefore, the complexity is $O(n)$. In the step of state merging, the system searches all pairs of most likely equivalent states. It takes $O(t^2)$, where t is the number of states in the PTA. In the worst-case scenario, t equals n (the number of messages), and we can approximate the complexity to be $O(n^2)$.

The progressive Needleman and Wunsch sequence alignment is conducted for clusters in the process of semantic deduction. A cluster consists of a set of messages having a similar structure. For the N&W algorithm, the complexity is $O(m^3l^2)$, where m is the average number of messages in a single cluster and l is the max length (bytes) of a single message. Overall, for all clusters, we can do the sequence alignment in parallel to reduce the processing time to $O(nm^2l^2)$.

Finally, in order to find the message dependencies, the system has to compute the Pearson correlation coefficient for every possible pair

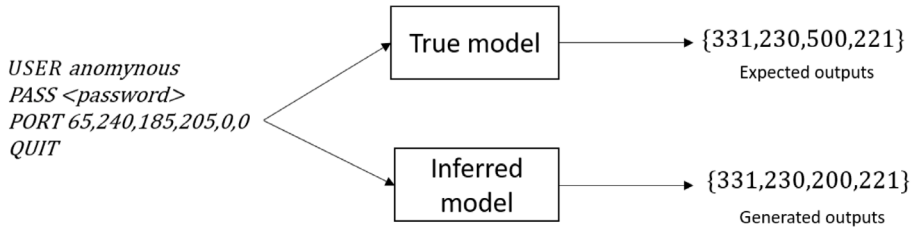

Fig. 8. An example of behavioral accuracy score.

Table 4
The processing time (minute) of the tasks for different protocol packet traces. The experiment is conducted in a workstation consists of an Intel Core™ i5-3210M dual-core CPU with 8 Gbyte of system memory. The summary of datasets is shown in Table 6.

Tasks/Processing Time (minutes)	FTP	SMTP	BitTorrent	PPLive
Data Preparation	4	1	1	3
Message Type Identification	32	10	10	18
FSM Construction	12	2	2	4
Semantic Deduction	50	12	18	20
Total	98	25	31	48

of fields. The time complexity is approximately $O(d^2)$, where d is the total number of possible fields. In the worst-case scenario, d equals to nl.

Let us take the processing of the FTP protocol trace as an example. There are a total of 200,000 messages (n) distributed in 27 clusters. The max length of a single message (l) is 40 bytes. Therefore, the number of messages in each cluster (m) is approximately 7400. The total number of possible fields (d) is approximately 80. Table 4 lists the processing time of each major task for different protocol packet traces. The most time-consuming task is the progressive N&W sequence alignment in the semantic deduction module.

6. Results and discussions

In this section, we present our evaluation results for FTP, SMTP, BitTorrent, and PPLive protocols. Fig. 9 represents the inferred extended finite state machines (EFSMs). As mentioned before, the quality of message type identification has a major impact on the inferred models so that we analyze it first before discussing the quality of inferred EFSMs.

6.1. Message type identification accuracy

The proposed ReFSM detected all of 26 FTP commands on the server-side. The remaining commands, a total of 33, according to those in RFC 959, could not be extracted because of the absence of traces. A total of nine SMTP keywords were detected. Table 5 shows the lists of keywords extracted from the network traces in human-readable form instead of the byte sequence. In the case of BitTorrent and PPLive protocol, the keywords are given in the form of byte sequences. For example, the first sequence {0 × 13, 0 × 42, 0 × 69, 0 × 74, 0 × 54, 0 × 6f, 0 × 72, 0 × 72, 0 × 65, 0 × 6e, 0 × 74, 0 × 20, 0 × 70, 0 × 72, 0 × 6f, 0 × 63, 0 × 6c} is marked as keyword HANDSHAKE in the BitTorrent protocol (The BitTorrent Protocol Specification, 2017). Except for the HANDSHAKE messages, the other keywords always start with a four-byte sequence and one following byte identifying the message type. For the PPLive, ReFSM identified eight clusters corresponding to eight keywords, as shown in Table 5. The third byte of each keyword reveals the type of certain message. This result is comparable to protocol state messages of PPLive in Veritas (Wang et al., 2011). The keywords analysis result of SMTP leads to an ideal precision score. However, the recall was 97% because two isolated clusters of EHLO and HELO messages were miss-classified from the same cluster.

By contrast, the messages of BITFIELD and REQUEST in BitTorrent protocol were similar. The proposed method was unable to find the keywords to distinguish and merge them into one cluster. Thus, the precision was only 94%, while recall was nearly perfect.

For all protocols, as shown in Fig. 10, values of pairwise precision and recall of ReFSM were higher than those of AutoReEngine. The iterative keyword analysis was helpful in finding the missing keywords of FTP; the K-means clustering algorithm helped calibrate the issue of over-specific extracted keywords. Thus, both precision and recall values of FTP were considerably improved. Similarly, the results of PPLive were higher than those in the AutoReEngine approach. For SMTP and BitTorrent, ReFSM remained of the same accuracy as that for AutoReEngine.

6.2. Quality of inferred EFSM

In this section, we analyze the quality of the inferred EFSM. First, we present the discussions on the effectiveness of using data guard and examine the impact of K-Tail parameters on the conciseness of final EFSM, and we then discuss the correctness, coverage scores, and behavioral accuracy score.

Table 3 summarizes the number of states of our EFSM for each protocol dataset with the K-Tail parameter. The EFSM models with our ReFSM with K-Tail of one are concise at all protocols. The number of states in the ideal models of FTP, SMTP, BitTorrent, and PPLive are 5, 8, 5 and 9. The K-Tail merging mechanism significantly reduces the number of states, and the K-Tail parameter impacts only on the conciseness and not on the quality of the inferred models. Both inferred models of K-Tail revealed similar behaviors that are analyzed in the next section.

6.2.1. Effectiveness of data guard

With the help of Daikon and Pearson product-moment correlation, ReFSM can infer the constraints on the value of fields and the value of fields across the messages. This constraint can be a form of data guards that check whether the field's value is valid.

Take the FTP as an example; the TYPE messages have the format "TYPE x," and the valid set of x is the enumerable set of {A, I}. If the value of x is not in {'A', 'I'} the incoming message will be rejected, and the return messages will be 500. If x is on the list, the protocol model will reply message with a code of 200. Another significant inferred data guard of FTP is in the PORT messages, with the format *port* $f_1, f_2, f_3, f_4, f_5, f_6$, and the valid set of f_i is between 0 and 255. It means the data guards will check whether f_i is in the range from 0 to 255. If not, it will reject the incoming message and return a message 500 code.

Furthermore, ReFSM uses the Pearson product-moments to find the pair of fields/variables that have a strong correlation; it helps to find the dependencies between fields. In the case of PORT messages of FTP, ReFSM found that f_1, f_2, f_3, f_4 have to be equal to the IPv4 address of servers. The data guards of this state will compare f_1, f_2, f_3, and f_4 to the server's IPv4 address.

In the case of SMTP, ReFSM found that the valid value of arguments after "MAIL FROM" or "MAIL TO/RCPT TO" have to contain an "@"

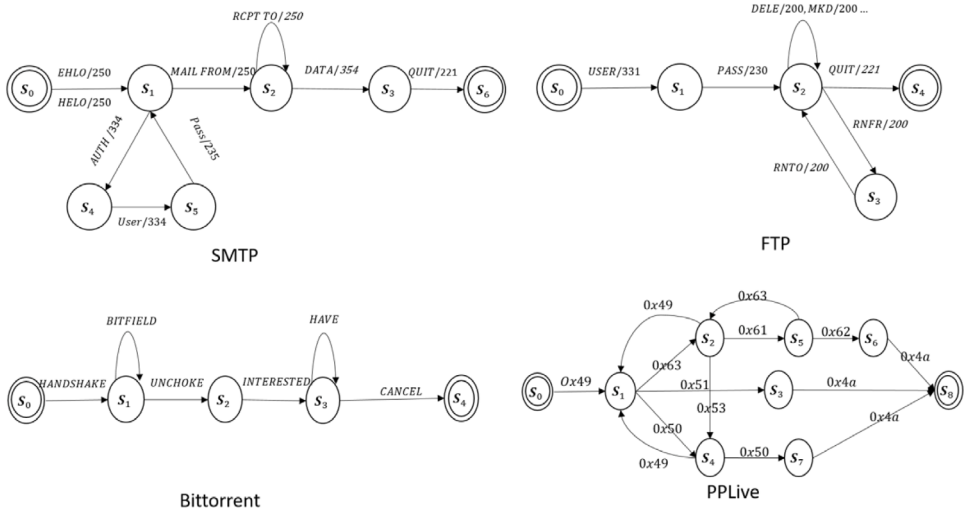

Fig. 9. The inferred extended finite state machines (EFSMs) for the protocols of FTP, SMTP, PPLive and BitTorrent.

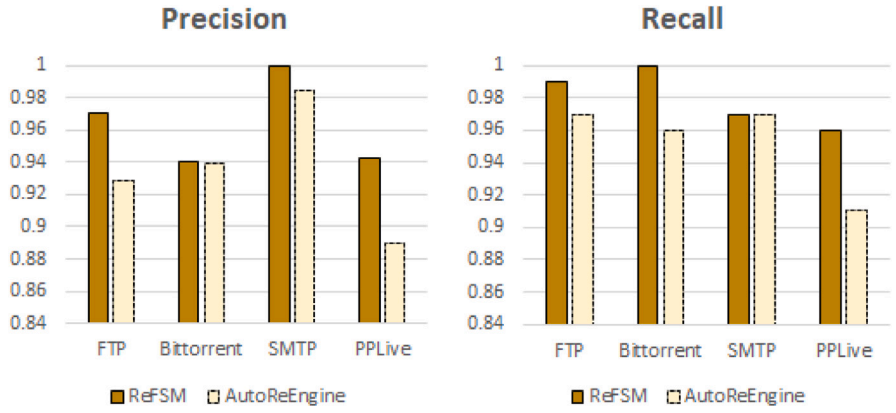

Fig. 10. The precision and recall results of message type identification.

Table 5
The results of keyword analysis on the protocols of FTP, SMTP, PPLive, and BitTorrent.

Protocol	Extracted keywords	#clusters/#true clusters
FTP	USER, PASS, PORT, OPTS, MODE, CWD, PASV, RETR, TYPE, HELP, SITE, STOR, SIZE, MKD, CLNT, ALLO, REST, STAT, MDTM, NLST, LIST, RNFR, RNTO, DELE, ABOR, QUIT	27/26
SMTP	EHLO, HELO, MAIL FROM: , RCPT TO: , DATA, QUIT, RSET, AUTH, User, Pass	10/9
BitTorrent	{0 × 13, 0 × 42, 0 × 69, 0 × 74, 0 × 54, 0 × 6f, 0 × 72, 0 × 72, 0 × 65, 0 × 6e, 0 × 74, 0 × 20, 0 × 70, 0 × 72, 0 × 6f, 0 × 63, 0 × 6c} (HANDSHAKE), {0 × 00, 0 × 00, 0 × 00, 0 × 01, 0 × 00} (CHOKE), {0 × 00, 0 × 00, 0 × 00, 0 × 01, 0 × 01} (UNCHOKE), {0 × 00, 0 × 00, 0 × 00, 0 × 01, 0 × 02} (INTERESTED), {0 × 00, 0 × 00, 0 × 00, 0 × 01, 0 × 03} (NOT INTERESTED) {0 × 00, 0 × 00, 0 × 00, 0 × 05, 0 × 04} (HAVE), {0 × 00, 0 × 00, 0 × 00, 0 × 01, 0 × 05} (BITFIELD), {0 × 00, 0 × 00, 0 × 00, 0 × 0d, 0 × 06} (REQUEST), {0 × 00, 0 × 00, 0 × 00, 0 × 0d, 0 × 08} (CANCEL)	9/10
PPLive	{0xe9, 0 × 03, 0 × 49, 0 × 01, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 98} {0xe9, 0 × 03, 0 × 4a, 0 × 01, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 01} {0xe9, 0 × 03, 0 × 50, 0 × 00, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 9b} {0xe9, 0 × 03, 0 × 51, 0 × 01, 0 × 98, 0xab} {0xe9, 0 × 03, 0 × 53, 0 × 00, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 5b} {0xe9, 0 × 03, 0 × 61, 0 × 01, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 01} {0xe9, 0 × 03, 0 × 62, 0 × 01, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 01} {0xe9, 0 × 03, 0 × 63, 0 × 01, 0 × 98, 0xab, 0 × 01, 0 × 02, 0 × 01}	9/8

such as "MAIL FROM x@y," where x and y are strings with at least one character (not empty string).

In BitTorrent protocol, after a client sends the first HANDSHAKE messages (the message that starts with the keywords mentioned in

Table 6
Summary of immediate results of EFSM inference for the protocols of FTP, SMTP, PPLive and BitTorrent.

Protocol	No. of Sessions	No. of PTA nodes	K-Tail	No. of States	No. of Transitions	No. of Data guards
	1000	9168	k=1	5	28	
			k=2	782	867	
FTP	750	8552	k=1	5	28	35
			k=2	650	760	
	500	4958	k=1	5	28	
			k=2	470	512	
SMTP	426	45	k=1	8	8	3
			k=2	45	64	
BitTorrent	260	32	k=1	5	10	5
			k=2	32	48	
PPLive	281	82	k=1	9	14	0
			k=2	92	82	

Section 6.1), the EFSM will hold 20 following bits as memories. When a client receives the response messages from servers, the data guards will compare the string of 20 bits in response messages with the value in memories of the state that were previously sent before transiting the states.

6.2.2. Coverage and correctness of inferred EFSMs

The parameter of k-folds equal to five is applied to assess coverage and correctness. It means that 20% of sessions are held in traces to test. The ratio of accepted and rejected is 4:1. The results, shown in Fig. 11, confirm the quality of our FSM part of EFSM. Almost 100% of the states in the specification are covered. This is mainly due to the simplicity of SMTP and the richness of the dataset. Also, all valid sessions are accepted by the inferred model.

Similarly, the coverage of BitTorrent and PPLive is also high. The cognitive complexity of FTP with a broad set of commands leads to the lowest coverage at 91%. There are some commands (transitions) in the test set that are randomly neglected in the training set so that the ReFSM is not able to learn. The correctness score is never below 90%, which guarantees that the models inferred are close to the actual models.

6.2.3. Behavioral accuracy

In the same way, the k-folds value of five was used to calculate the behavioral accuracy. Primarily, the ReFSM relies on Daikon to generate the data guard, including the valid syntax of data and the dependencies. Table 6 lists the number of data constraints extracted by ReFSM from datasets of four protocols. Among many data constraints of FTP, the constraints on the PORT command's arguments are significant. The first four numbers, separated by a colon, should be mapped to the IP address of the server and the port number should be higher than 255. One of the detected data guards of BitTorrent is that the first 20 bytes of *hash_id* should be consistent with a HANDSHAKE message.

To show the effectiveness of the extended part (data flow) of EFSM, comparative study of behavioral accuracy with FSM only was carried out. Basically, FSMs of existing works only represent control flow so that the outputs are determined only by the input message type and without considering the data flow portion. For example, with the incoming message of "PORT 65,240,180,205,0,0," shown in Fig. 8, the FSMs always produced the response code "200" (to notify successful action), based on the recognition of PORT command. At the same time, the EFSMs, with the help of the data guard, check the data of "65,240,180,205,0,0" and return the code "500" to notify unsuccessful action.

For FTP protocol, shown on the right-hand side of Fig. 11, 89% on average of generated outputs of inferred EFSMs match the expected outputs. The FSM without data guard and memories only correctly produced 78% of outputs. It means that the behavior of the inferred model is nearly comparable to the actual model in each state. The remaining errors are related to the information that is absent in traces. For instance, with a received message of "SIZE reference.tar.gz," the inferred EFSM outputs the message code of "213" and the size of the file. However, the actual reply message is "550" because there is no such file in this system. This characteristic depends on the environment of the system and cannot be inferred from network traces. The results of 93% and 98% indicate that ReFSM can construct faithful EFSM of the corresponding BitTorrent and SMTP protocols. The ReFSM can adequately expose the same behaviors to the actual model manually derived from the specification.

The design of PPlive is based on two protocols: channels list distribution protocol and streaming protocols. The *channels list distribution protocol* relies on the HTTP protocol to send a zip file from a server to a client. This zip file consists of many records of TV channels and peers watching the channels. After unzipping the file and obtaining peer-related information from specific channels, PPLive client uses the *streaming protocol* to register itself to the overlay network and start to exchange the video chunk. The protocol adopts an encryption mechanism to prevent it from being identified and supervised by the ISP. Therefore, our method fails for this protocol as other statistic approach methods do and the behavioral accuracy of PPLive is not provided.

6.2.4. Impact of trace size

In this work, the metrics of coverage and correctness are used to demonstrate the proposed EFSM reconstruction is at least equivalent or better than those in other works. Coverage is calculated by the percentage of actual sessions accepted by the inferred model. Correctness is the percentage of inferred sessions that are genuinely accepted by the true model, where the acceptance is defined as the last state's reachability.

Therefore, in the training phase, the more sessions in a network trace, the more faithful a model can be obtained. However, it leads to an increase in processing time for learning. To evaluate the system performance, files of FTP traces with different sizes are fed into ReFSM to infer the EFSM models and compute the correctness, coverage, and behavioral scores. The size of the training dataset was increased by 10% in each run. Fig. 12 shows the relationship between the number of sessions and the quality of the models. The ideal number of sessions for the EFSM model inference is approximately 400.

7. Conclusion

In this work, we propose the ReFSM, a novel analysis methodology that infers the behavioral model of protocols in the form of an extended finite state machine (EFSMs) from network traces. In particular, the *data flow*, represented by the *data guard* and *memory* on the transitions of states, is taken into account for the construction of the EFSM. It is different from the previous work on traditional reverse engineering problems where only the *control flow* information is considered. The inferred models are equipped with the constraints on data values. Daikon deduces those constraints using correlation analysis techniques. Furthermore, ReFSM leverages the K-Tail mechanism, which is widely

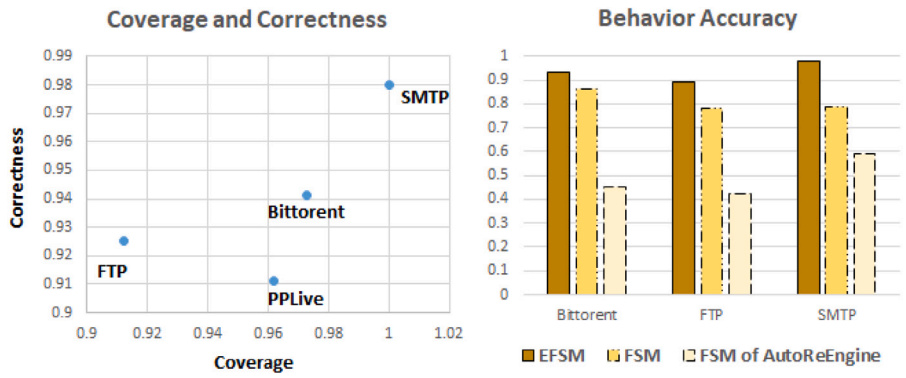

Fig. 11. The coverage, correctness and behavioral accuracy of inferred EFSMs. Coverage is calculated by the percentage of actual sessions (generated by the true model) accepted by the inferred model. Correctness is the percentage of inferred sessions (accepted by the inferred model) that are genuinely accepted by the true model.

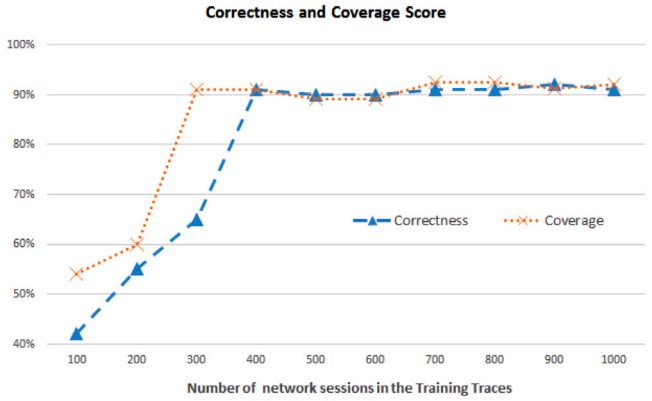

Fig. 12. The score of correctness and coverage represents the quality of an inferred model based on the number of training traces. Coverage is calculated by the percentage of actual sessions accepted by the inferred model. Correctness is the percentage of inferred sessions that are genuinely accepted by the true model. The ideal number of sessions for EFSM model inference is approximately 400 as the correctness score grows up to 90%.

used in software behavior inference to enhance the accuracy of classical FSM (control flow) operations.

A prototype was developed based on Java language and evaluated with network traffic traces of four protocols: FTP, SMTP, PPLive, and BitTorrent. Several performance metrics were also used to evaluate the results. Since the message type identification is defined as an unsupervised clustering process, a pair-wise precision and recall score was adopted. For the quality of inferred models, the correctness and coverage were used to evaluate the part of classical FSM. The behavioral accuracy was measured for the evaluation of the EFSM.

The preliminary experimental results indicate the excellent quality of the inferred EFSMs. For all the protocols tested, BitTorrent, FTP and SMTP, the proposed EFSM reached a two-fold increase in behavioral accuracy compared to that of the FSM of AutoReEngine. Leverage on the iterative keyword analysis and k-mean clustering, the precision and recall values of message type identification are, at least, well above 94% and 96%, respectively. Moreover, armed with the extra data guards, the scores of coverage and correctness attained 93% and 98% for BitTorrent and SMTP.

The high scores of pair-wise precision and recall also represent the effectiveness of our hybrid method for messages clustering. The messages are correctly identified and grouped based on the specifications required. This accurate result is the premise on which to construct the behavioral models. It guarantees that at least ReFSM is capable of inferring the corrected FSMs. Based on the excellent behavioral accuracy scores, the ReFSM is capable of capturing the behavior of incoming messages for a given protocol. It provides the model with output response messages to enable network systems administrators to conduct network testing and manage security operations more effectively.

7.1. Limitations

There are four main limitations associated with the design of ReFSM: traces dependency, key–value pair protocols, encrypted protocols and a pre-defined set of data guard forms. As the construction of the behavioral model depends on the network traffic traces, the proposed inference method cannot identify the message type if no particular messages of this type appear in the traces. Therefore, it is difficult to infer a complete model of the protocol if the individual states cannot be reached as a result of an incomplete collection of messages in the traces.

The encryption process changes the statistical properties of the protocol so that the proposed inference methods cannot accurately extract the keywords and identify the message types. Therefore, ReFSM cannot infer the behavioral model of encrypted protocols. ReFSM leverages the N&W sequence alignment algorithm to extract the fields and format of messages before performing the semantic deduction. In practice, this method is efficient in HTTP and FTP protocols where payloads are encoded in the form of a key–value pair. It is also suitable for protocols, such as BitTorrent, where each byte is implied as a field. Another limitation is associated with using the Daikon algorithm to infer the data guards in the protocol message. The essence of the data guard inference is to formulate a function which maps data value into a Boolean value of true or false. Daikon can only infer limited types of constraints such as *range*, *constant*, and *enumerative* set of values. It is, therefore, a challenging task to find all possible data guards for all protocols.

7.2. Future work

We are currently working on a comparative study with the existing works to verify our method. In particular, the proposed methodology is applied to test case generation tools and stateful fuzzing applications to assess the usefulness of our inferred EFSMs. Some enhancements regarding the prototype of test case generation, based on the inferred EFSMs model with the unique input/output (UIO) (Dahbura et al., 1990) sequence, can be pursued to showcase such capability. Moreover, Walkinshaw et al. (2016) proposed the machine learning techniques to infer data guard instead of using Daikon in the ReFSM. Data guard is in the form of a Decision Tree binary classifier that constructed by C4.5 algorithm. The techniques provided by Foster et al. (2018) can also be adopted to build more efficient and accurate EFSM.

In addition to the N&W sequence alignment algorithm, readers are recommended to pursue the adoption of methods (Cui et al., 2007; Bermudez et al., 2016) which are able to carry out format extractions

with higher accuracy for more protocols. The machine learning classifier (Walkinshaw et al., 2016) producing binary output labels can also be utilized as a potentially improved solution for the construction of data guards.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded in part by the Ministry of Science and Technology (MoST), Taiwan, and in part by Estinet Technologies Inc. and Chunghua Telecom.

References

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases. In: VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 487–499, http://dl.acm.org/citation.cfm?id=645920.672836.

Amiri, F., Yousefi, M.R., Lucas, C., Shakery, A., Yazdani, N., 2011. Mutual information-based feature selection for intrusion detection systems. J. Netw. Comput. Appl. 34 (4), 1184–1199. http://dx.doi.org/10.1016/j.jnca.2011.01.002, Advanced Topics in Cloud Computing. http://www.sciencedirect.com/science/article/pii/S1084804511000038.

Antunes, J., Neves, N., Verissimo, P., 2011. Reverse engineering of protocols from network traces. In: 2011 18th Working Conference on Reverse Engineering. pp. 169–178. http://dx.doi.org/10.1109/WCRE.2011.28.

Beddoe, M.A., 2018. Network protocol analysis using bioinformatics algorithms. http://www.4tphi.net/~awalters/PI/PI.html. (Accessed 04 January 2018).

Bermudez, I., Tongaonkar, A., Iliofotou, M., Mellia, M., Munaf, M.M., 2016. Towards automatic protocol field inference. Comput. Commun. 84 (C), 40–51. http://dx.doi.org/10.1016/j.comcom.2016.02.015.

Biermann, A.W., Feldman, J.A., 1972. On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. C-21 (6), 592–597. http://dx.doi.org/10.1109/TC.1972.5009015.

Bossert, G., Guihery, F., Hiet, G., 2014. Towards automated protocol reverse engineering using semantic information. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '14. ACM Press, Kyoto, Japan, pp. 51–62. http://dx.doi.org/10.1145/2590296.2590346, http://dl.acm.org/citation.cfm?doid=2590296.2590346.

Cui, W., Kannan, J., Wang, H.J., 2007. Discoverer: Automatic protocol reverse engineering from network traces. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium. In: SS'07, USENIX Association, Berkeley, CA, USA, pp. 14:1–14:14, http://dl.acm.org/citation.cfm?id=1362903.1362917.

Dahbura, A., Sabnani, K., Uyar, M., 1990. Formal methods for generating protocol conformance test sequences. Proc. IEEE 78 (8), 1317–1326. http://dx.doi.org/10.1109/5.58319.

Dallmeier, V., Knopp, N., Mallon, C., Fraser, G., Hack, S., Zeller, A., 2012. Automatically generating test cases for specification mining. IEEE Trans. Softw. Eng. 38 (2), 243–257. http://dx.doi.org/10.1109/TSE.2011.105.

Duchene, J., Guernic, C.L., Alata, E., Nicomette, V., Kaaniche, M., 2018. State of the art of network protocol reverse engineering tools. J. Comput. Virol. Hacking Tech. 14 (1), 53–68. http://dx.doi.org/10.1007/s11416-016-0289-8.

Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G., 1998. Biological sequence analysis: Probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, http://dx.doi.org/10.1017/CBO9780511790492, http://ebooks.cambridge.org/ref/id/CBO9780511790492.

Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D., 2001. Dynamically discovering likely program invariants to support program evolution. IEEE Trans. Softw. Eng. 27 (2), 99–123. http://dx.doi.org/10.1109/32.908957.

Foster, M., Taylor, R.G., Brucker, A.D., Derrick, J., 2018. Formalising extended finite state machine transition merging. In: Sun, J., Sun, M. (Eds.), Formal Methods and Software Engineering, vol. 11232. Springer International Publishing, Cham, pp. 373–387, Series Title: Lecture Notes in Computer Science http://link.springer.com/10.1007/978-3-030-02450-5_22.

Goo, Y.-H., Shim, K.-S., Lee, M.-S., Kim, M.-S., 2019. Protocol specification extraction based on contiguous sequential pattern algorithm. IEEE Access http://dx.doi.org/10.1109/access.2019.2905353.

Hatzivassiloglou, V., McKeown, K.R., 1993. Towards the automatic identification of adjectival scales: clustering adjectives according to meaning. In: Proceedings of the 31st Annual Meeting on Association for Computational Linguistics. In: ACL '93, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 172–182. http://dx.doi.org/10.3115/981574.981597.

Jaccard, 1912. The distribution of the flora in the alpine zone. The New Phytologist, https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x.

Kleber, S., van der Heijden, R.W., Kargl, F., 2020. Message type identification of binary network protocols using continuous segment similarity. arXiv:2002.03391 [cs]. http://arxiv.org/abs/2002.03391.

Kleber, S., Maile, L., Kargl, F., 2019. Survey of protocol reverse engineering algorithms: decomposition of tools for static traffic analysis. IEEE Commun. Surv. Tutor. 21 (1), 526–561. http://dx.doi.org/10.1109/COMST.2018.2867544.

Kohavi, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. In: IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1137–1143, http://dl.acm.org/citation.cfm?id=1643031.1643047.

Krka, I., Brun, Y., Medvidovic, N., 2014. Automatic mining of specifications from invocation traces and method invariants. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. In: FSE 2014, Association for Computing Machinery, New York, NY, USA, pp. 178–189. http://dx.doi.org/10.1145/2635868.2635890.

Krueger, T., Gascon, H., Kramer, N., Rieck, K., 2012. Learning stateful models for network honeypots. In: Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence, AISec 12. ACM Press, p. 37. http://dx.doi.org/10.1145/2381896.2381904, http://dl.acm.org/citation.cfm?doid=2381896.2381904.

Leita, C., Mermoud, K., Dacier, M., 2005. ScriptGen: an automated script generation tool for honeyd. In: 21st Annual Computer Security Applications Conference (ACSAC'05). pp. 12 pp.–214. http://dx.doi.org/10.1109/CSAC.2005.49.

Lo, D., Mariani, L., Santoro, M., 2012. Learning extended FSA from software: An empirical assessment. J. Syst. Softw. 85 (9), 2063–2076. http://dx.doi.org/10.1016/j.jss.2012.04.001, https://doi.org/10.1016/j.jss.2012.04.001.

Lorenzoli, D., Mariani, L., Pezze, M., 2006. Inferring state-based behavior models. In: Proceedings of the 2006 International Workshop on Dynamic Systems Analysis. In: WODA '06, ACM, New York, NY, USA, pp. 25–32. http://dx.doi.org/10.1145/1138912.1138919.

Lorenzoli, D., Mariani, L., Pezzè, M., 2008. Automatic generation of software behavioral models. In: Proceedings of the 30th International Conference on Software Engineering. In: ICSE 08, Association for Computing Machinery, New York, NY, USA, pp. 501–510. http://dx.doi.org/10.1145/1368088.1368157.

Luo, J.-Z., Yu, S.-Z., 2013. Position-based automatic reverse engineering of network protocols. J. Netw. Comput. Appl. 36 (3), 1070–1077. http://dx.doi.org/10.1016/j.jnca.2013.01.013, http://www.sciencedirect.com/science/article/pii/S1084804513000222.

Mariani, L., Pastore, F., 2008. Automated identification of failure causes in system logs. In: 2008 19th International Symposium on Software Reliability Engineering (ISSRE). pp. 117–126. http://dx.doi.org/10.1109/ISSRE.2008.48.

Mariani, L., Pastore, F., Pezze, M., 2011. Dynamic analysis for diagnosing integration faults. IEEE Trans. Softw. Eng. 37 (4), 486–508. http://dx.doi.org/10.1109/TSE.2010.93.

Mariani, L., Pezzè, M., Santoro, M., 2017. GK-tail+ an efficient approach to learn software models. IEEE Trans. Softw. Eng. 43 (8), 715–738. http://dx.doi.org/10.1109/TSE.2016.2623623.

Narayan, J., Shukla, S.K., Clancy, T.C., 2015. A survey of automatic protocol reverse engineering tools. ACM Comput. Surv. 48 (3), 40:1–40:26. http://dx.doi.org/10.1145/2840724.

Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48 (3), 443–453. http://dx.doi.org/10.1016/0022-2836(70)90057-4, http://www.sciencedirect.com/science/article/pii/0022283670900574.

Pang, R., Paxson, V., 2003. A high-level programming environment for packet trace anonymization and transformation. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. In: SIGCOMM '03, ACM, New York, NY, USA, pp. 339–351. http://dx.doi.org/10.1145/863955.863994.

Paxson, V., 1999. Bro: A system for detecting network intruders in real-time. Comput. Netw. 31 (23–24), 2435–2463. http://dx.doi.org/10.1016/S1389-1286(99)00112-7, http://linkinghub.elsevier.com/retrieve/pii/S1389128699001127.

Petrenko, A., Boroday, S., Groz, R., 2004. Confirming configurations in EFSM testing. IEEE Trans. Softw. Eng. 30 (1), 29–42. http://dx.doi.org/10.1109/TSE.2004.1265734.

Postel, J., Reynolds, J., File Transfer Protocol. https://tools.ietf.org/html/rfc959.

Sija, B.D., Goo, Y.-H., Shim, K.-S., Hasanova, H., Kim, M.-S., 2018. A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view. Secur. Commun. Netw. http://dx.doi.org/10.1155/2018/8370341, https://www.hindawi.com/journals/scn/2018/8370341/.

Sokal, R.R., Michener, C.D., 1958. A statistical method for evaluating systematic relationships. Univ. Kansas Sci. Bull. 28, 1409–1438.

Srikant, R., Agrawal, R., 1996. Mining sequential patterns: Generalizations and performance improvements. In: Apers, P., Bouzeghoub, M., Gardarin, G. (Eds.), Advances in Database Technology — EDBT '96. In: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 1–17. http://dx.doi.org/10.1007/BFb0014140.

Tan, P.-N., Steinbach, M., Kumar, V., 2006. Introduction to Data Mining. Pearson, Boston.

Tappler, M., Aichernig, B.K., Bloem, R., 2017. Model-based testing IoT communication via active automata learning. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). pp. 276–287. http://dx.doi.org/10.1109/ICST.2017.32.

The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html.

Trifil, A., Burschka, S., Biersack, E., 2009. Traffic to protocol reverse engineering. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. pp. 1–8. http://dx.doi.org/10.1109/CISDA.2009.5356565.

Walkinshaw, N., Taylor, R., Derrick, J., 2016. Inferring extended finite state machine models from software executions. Empir. Softw. Eng. 21 (3), 811–853. http://dx.doi.org/10.1007/s10664-015-9367-7.

Wang, Y., Xiang, Y., Zhou, W., Yu, S., 2012. Generating regular expression signatures for network traffic classification in trusted network management. J. Netw. Comput. Appl. 35 (3), 992–1000. http://dx.doi.org/10.1016/j.jnca.2011.03.017, Special Issue on Trusted Computing and Communications. http://www.sciencedirect.com/science/article/pii/S1084804511000713.

Wang, Y., Zhang, Z., Yao, D.D., Qu, B., Guo, L., 2011. Inferring protocol state machine from network traces: A probabilistic approach. In: Lopez, J., Tsudik, G. (Eds.), Applied Cryptography and Network Security. In: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 1–18. http://dx.doi.org/10.1007/978-3-642-21554-4_1.

Ying-Dar Lin (F'93) is a Distinguished Professor of computer science at National Chiao Tung University (NCTU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose during 2007–2008, CEO at Telecom Technology Center, Taiwan, during 2010–2011, and Vice President of National Applied Research Labs (NARLabs), Taiwan, during 2017–2018. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and automated tools, and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, later acquired by D-Link Corp, and O'Prueba Inc. in 2018. His research interests include network security, wireless communications, and network softwarization. His work on multi-hop cellular was the first along this line, and has been cited over 850 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (2014–2017), ONF Research Associate, and received in 2017 Research Excellence Award and K. T. Li Breakthrough Award. He has served or is serving on the editorial boards of several IEEE journals and magazines, and is the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST). He published a textbook, Computer Networks: An Open Source Approach (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Yu-Kuen Lai (S'03-M'06-SM'14) received the M.S. and Ph.D. degrees in electrical and computer engineering from North Carolina State University (NCSU) at Raleigh, NC, USA in 1997 and 2006, respectively. He is an Associate Professor with the Electrical Engineering Department, Chung-Yuan Christian University (CYCU), Chung-Li, Taiwan. From 1997 to 2002, he was a Senior ASIC Design Engineer with Delta Networks, Inc., and Applied Micro Circuit Corporation (AMCC) at Research Triangle Park, NC, USA. His research interests include network processor architecture, streaming data processing, network traffic analysis, FPGA systems design, and computer network security. Dr. Lai served as the Electronic Communication Officer for the IEEE Taipei Section in 2015. He was the recipient of the CYCU Distinguished Teaching Award and CYCU Outstanding Teaching Award in 2011 and 2017, respectively. He is currently the director of the Information Technology Division at C.Y. Chang Memorial Library at CYCU.

Quan Tien Bui received his M.S. degree in Computer Science from National Chiao Tung University, Taiwan in 2018. His research interests include network security, IoT and software reverse engineering.

Yuan-Cheng Lai (A'05-M'13) received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1997. In August 1998, he joined the faculty of the Department of Computer Science and Information Science, National Cheng Kung University, Tainan, Taiwan. In August 2001, he joined the faculty of the Department of Information Management, National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan, where he has been a Professor since February 2008. His research interests include performance analysis, protocol design, wireless networks, and web-based applications.