Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Active inference of protocol state machines from incomplete message domains*#

Maohua GUO, Yuefei ZHU, Jinlong FEI‡

Key Laboratory of Cyberspace Security, Ministry of Education, Zhengzhou 450001, China

E-mail: czxing.2019@outlook.com; yfzhu17@sina.com; feijinlong_2021@163.com

Received June 6, 2024; Revision accepted Sept. 6, 2024; Crosschecked Oct. 30, 2025; Published online Dec. 12, 2025

Abstract: Inferring protocol state machines from observable information presents a significant challenge in protocol reverse engineering (PRE), especially when passively collected traffic suffers from message loss, resulting in an incomplete protocol state space. This paper introduces an innovative method for actively inferring protocol state machines using the minimally adequate teacher (MAT) framework. By incorporating session completion and deterministic mutation techniques, this method broadens the range of protocol messages, thereby constructing a more comprehensive input space for the protocol state machine from an incomplete message domain. Additionally, the efficiency of active inference is improved through several optimizations for the L_{M}^{+} algorithm, including traffic deduplication, the construction of an expanded prefix tree acceptor (EPTA), query optimization based on responses, and random counterexample generation. Experiments on the real-time streaming protocol (RTSP) and simple mail transfer protocol (SMTP), which use Live555 and Exim implementations across multiple versions, demonstrate that this method yields more comprehensive protocol state machines with enhanced execution efficiency. Compared to the L_{M}^{+} algorithm implemented by AALpy, Act_Infer achieves an average reduction of approximately 40.7% in execution time and significantly reduces the number of connections and interactions by approximately 28.6% and 46.6%, respectively.

Key words: Protocol reverse engineering (PRE); Protocol state machine; Active inference; Incomplete message domains; Input space

https://doi.org/10.1631/FITEE.2400487 **CLC number:** TP393

1 Introduction

In complex network environments, ensuring precise and stable data transmission between communication entities relies on standardizing their interactive behaviors through network protocols. The request for comments (RFC, https://www.rfc-editor.org/), sponsored and published by the Internet Society (ISOC), serves as the comprehensive repository for nearly all public Internet standards, encompassing 9565 protocol specification documents as of March 2024. However, the plethora of botnets, Trojans, and malware employing proprietary protocols for communication (Chandler, 2023) poses significant challenges to network security oversight, driving an increased focus on protocol reverse engineering (PRE) in fields such as intrusion detection (Abdulganiyu et al., 2023; Saied et al., 2024), vulnerability mining (Pham et al., 2020; Natella, 2022; Yu et al., 2024), and malware analysis (Antonakakis et al., 2017; de Carli et al., 2017).

PRE (Huang et al., 2022) can be broadly classified into two implementation approaches. The first approach involves analyzing program instructions (Ma

‡ Corresponding author

* Project supported by the Key JCJQ Program of China (Nos. 2020-JCJQ-ZD-021-00 and 2020-JCJQ-ZD-024-12)

Electronic supplementary materials: The online version of this article (https://doi.org/10.1631/FITEE.2400487) contains supplementary materials, which are available to authorized users

ORCID: Maohua GUO, https://orcid.org/0000-0001-7990-4165; Yuefei ZHU, https://orcid.org/0000-0002-9559-8783; Jinlong FEI, https://orcid.org/0000-0001-8499-9402

© Zhejiang University Press 2025

et al., 2022), using dynamic taint analysis or symbolic execution techniques for high-precision reverse engineering. However, obstacles such as protection mechanisms, obfuscated binaries, or packaged files often hinder dynamic debugging and the analysis of protocol entities. The second approach centers on analyzing message sequences (Li et al., 2023), typically spanning preprocessing, protocol format inference, and protocol state machine inference stages. Preprocessing lays the groundwork for reverse engineering, including tasks like data clustering (Le et al., 2024). Common clustering algorithms include the nearest neighbor clustering (*K*-means), unweighted pair-group method with arithmetic means (UPGMA), and density-based spatial clustering of applications with noise (DBSCAN). Protocol format inference is accomplished through static traffic analysis to segment protocol messages (Wang YP et al., 2012; Kleber et al., 2018, 2022; Sun et al., 2019; Chandler et al., 2023), infer protocol keywords (Ye et al., 2021; Chandler, 2023; Tang et al., 2023), and perform semantic analysis (Bermudez et al., 2016; Wang XW et al., 2020; Wang YP et al., 2022). At this stage, the protocol is typically categorized as either a text-based protocol or a binary-based protocol based on whether the messages can be converted into JSON, XML, or other types of textual files. The detailed analysis of the protocol format lays a foundation for the inference stage of the protocol state machine. Protocol state machine inference (Székely et al., 2021; Sun et al., 2022) consolidates all known information and analysis results, employing state machines to formalize and regularize protocol interactive behavior. The research presented in this paper is based on static execution traces. While numerous solutions have been proposed for protocol format inference, there is a notable gap in research on protocol state machine inference. Analyzing unknown protocol state machines can significantly assist security analysts in monitoring and verifying protocol state transitions.

1.1 Motivation

There are two primary approaches to protocol state machine inference based on message sequences, depending on the interaction with the target protocol entity: passive inference (Leita et al., 2005; Shevertalov and Mancoridis, 2007; Whalen et al., 2010; Antunes et al., 2011; Wang YP et al., 2011; Lee et al., 2018; Lin et al., 2020; Sun et al., 2022) and active inference (Cho et al., 2010; Bossert et al., 2014; Wang C et al., 2015; Székely et al., 2021; Pan et al., 2023).

Passive inference uses collected offline traffic to extract type and state-related information from messages. It employs directed graphs (Shevertalov and Mancoridis, 2007), Markov models (Whalen et al., 2010), prefix tree acceptors (PTA) (Antunes et al., 2011; Wang YP et al., 2011; Lee et al., 2018; Lin et al., 2020), and random protocol converters (Sun et al., 2022) to describe protocol state transitions. By designing compatible state-merging algorithms, passive inference simplifies the state machine. This lightweight method is invaluable for quickly reconstructing protocol state machines using known data. However, passive inference has inherent limitations. (1) Lack of counterexample samples: Offline traffic generally lacks counterexample samples, making it challenging to obtain an accurate minimal state machine. Learning with only positive examples is an NP-complete problem (Gold, 1967). (2) State explosion: A large volume of interactive data can cause the number of states in the protocol state machine model to explode. State-merging strategies may lead to model generalization issues.

Active inference relies on active learning algorithms for automata. After thoroughly analyzing the message format, it uses collected offline data or constructs new protocol-compliant packets, sends queries to the protocol entity, and records response results for repeated inference. This method can theoretically produce the smallest and most complete protocol state machine. Common algorithms include the L^* algorithm by Angluin (1987) and the L^*_{M} and L^+_{M} algorithms for Mealy machines by Shahbaz (2008). However, active learning of protocol state machines has two key prerequisites: the basic input space must be known and the protocol entity must be reset before each query. Despite its potential, active inference faces the following practical challenges. (1) Incomplete input space: Due to incomplete sessions, missing abnormal behavior messages, and incomplete coverage of protocol message types in offline data, it is challenging to directly obtain a complete protocol input space from an incomplete message domain. (2) High overhead: The complexity of state transitions in real-world protocols necessitates many query

requests to the protocol entity. The need to reset the protocol entity before each query and the latency during the interaction process can lead to significant overhead.

1.2 Contributions of this paper

In addressing the challenges faced in current research, we optimize previous findings by expanding the input space and reducing query costs. We propose an active inference method for protocol state machines based on incomplete message domains, termed Act_Infer. Our contributions are highlighted in the following three areas.

1. Comprehensive input space expansion: We leverage collected offline data as a foundation and use proactive interaction to complement missing messages in sessions. Additionally, deterministic mutation techniques are employed to expand the variety of missing messages. This approach aims to construct as complete an input space for the protocol state machine as possible from an initially incomplete message domain.

2. Efficiency enhancements via the minimally adequate teacher (MAT) framework: Building on the MAT framework, we develop a mapper to replay offline traffic, facilitating proactive interaction with protocol entities. This enhances the efficiency of active protocol state machine inference through several key optimizations, including traffic deduplication—reduces redundant data to streamline the inference process, expanded PTA (EPTA)—constructs a more efficient and manageable protocol state machine structure, query optimization—improves the efficiency of queries based on the responses received, and random counterexample generation—generates random counterexamples based on state transitions to refine and improve the learning process.

3. Practical application and evaluation: We apply Act_Infer to multiple versions of the real-time streaming protocol (RTSP) and simple mail transfer protocol (SMTP), specifically Live555 (http://live555.com) and Exim (https://www.exim.org) implementations. We evaluate the method's execution efficiency in terms of execution time and number of queries, connections, and interactions. The results demonstrate a significant improvement over previous research outcomes.

2 Related works

This section reviews the current research on protocol state machine inference based on message sequences; the analytical methods employed and their principal contributions are summarized and compiled in Table 1. This section discusses the limitations of prior research, particularly those related to active inference methods.

2.1 Passive inference

ScriptGen by Leita et al. (2005) pioneered the inference of protocol state machines from message sequences. By setting maximum out-degree and state thresholds, they established Moore-type state machines for clients and servers. PEXT (Shevertalov and Mancoridis, 2007) used the longest common substring algorithm to merge states and construct sequential models based on message sequences, which diverged from traditional protocol state machines. Antunes et al. (2011) first used a PTA structure to construct protocol state machines, recording state transition frequencies and applying minimization algorithms for Moore state machines to merge equivalent states. However, these methods modeled protocol behavior using finite state acceptors (FSAs), lacking request and response relationships due to one-way message analysis.

Recent methods like PRETT (Lee et al., 2018) and ReFSM (Lin et al., 2020) expanded upon tree structures by considering bidirectional message interactions, extending traditional protocol state machines to build finite state transducers (FSTs), specifically Mealy state machines.

Additionally, some researchers incorporated probabilistic information into state transitions. Whalen et al. (2010) introduced a Markov model to generate protocol state machines with probabilistic transitions, and Wang YP et al. (2011) proposed a probabilistic protocol state machine (P-PSM). These methods, however, overlooked the interactive information between protocols. To address this, Sun et al. (2022) proposed the Sptia-PL method, combining interactive protocol information with state transition probabilities and optimizing for state explosion and model generalization issues. Nevertheless, incorporating statistical information from different offline data distributions into models results in unreliable transition probabilities between protocol states.

2.2 Active inference

Active inference typically models protocol behavior using Mealy machines, employing the MAT framework proposed by Angluin (1987). This framework consists of a learner and an oracle. The learner, aware only of the input/output alphabet of the system under learning (SUL), actively queries the oracle, which has complete information about the SUL, to learn the model.

As depicted in Fig. 1, in the practical inference of a protocol state machine, a mapper translates between protocol messages and abstract input/output sequences. The learner issues membership queries, sending a sequence of queries based on the alphabet to the oracle. The oracle forwards this sequence to the mapper, translating it into specific protocol request messages and sending them to the SUL. The SUL responds according to its current state, and the mapper converts the output sequence, sending it back to the oracle, which feeds it back to the learner. After a series of queries, the learner constructs a hypothesis state machine that matches the SUL's behavior. This hypothesis is verified through equivalence queries, and if incorrect, a counterexample is provided, iterating the learning process until the correct state machine is obtained.

Dispatcher by Cho et al. (2010) was the first to apply the L^* algorithm to infer network protocol state machines, initially reversing the Mega-D botnet's message format and manually abstracting message formats into types for model learning. They improved the L^*

Table 1 Main research contributions of protocol state machine inference based on message sequences

Method (reference)	Algorithm	Model (style)	Passive	Active	Probability	Variability	Generalization
ScriptGen (Leita et al., 2005)	–	FSA (Moore)	√				
PEXT (Shevertalov and Mancoridis, 2007)	LCS	FSA (Moore)	√				
Whalen et al. (2010)'s	HMM	PFSA (Moore)	√		√		
Dispatcher (Cho et al., 2010)	L^*	(Mealy)		√			
ReverX (Antunes et al., 2011)	PTA	FSA (Moore)	√				√
Veritas (Wang YP et al., 2011)	PTA	PFSA (Moore)	√		√		√
Netzob (Bossert et al., 2014)	L^*	(Mealy)		√			
Wang C et al. (2015)'s	EPTA+L^+_M	(Mealy)	√	√		√	√
PRETT (Lee et al., 2018)	PTA	FST (Mealy)	√				
ReFSM (Lin et al., 2020)	PTA	FST (Mealy)	√		√		√
Székely et al. (2021)'s	L^+_M	(Mealy)		√			√
Sptia-PL (Sun et al., 2022)	SPT	FST (Mealy)	√		√		√
Pan et al. (2023)'s	L^+_M	(Mealy)		√		√	√

LCS: longest common subsequence; HMM: hidden Markov model; SPT: stochastic protocol finite-state transducer; PFSA: probabilistic FSA

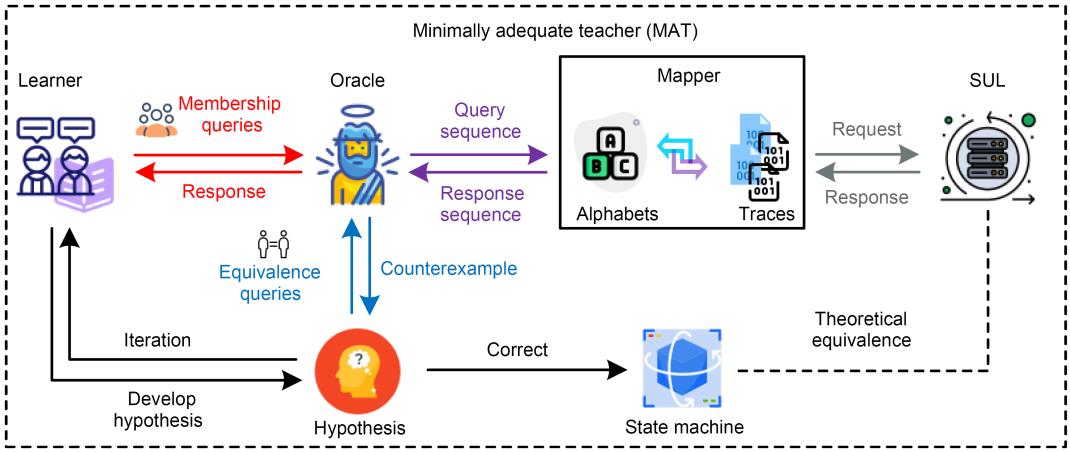

Fig. 1 Overview of the active inference process for protocol state machines

algorithm by incorporating a caching mechanism to reduce redundant queries, a strategy adopted by the open-source tool Netzob (Bossert et al., 2014). Wang C et al. (2015) integrated passive and active inference, using existing data to construct an EPTA, reducing the number of queries. They proposed a strategy based on mutating positive sample variants to accelerate counterexample discovery, although this approach did not expand the protocol's input space.

Székely et al. (2021) advanced the L_{M}^{+} algorithm by addressing the issue of excessively large input/output alphabets. They refined the mapping relationship between protocol messages and the abstract alphabet, introducing the concept of message subtypes. This allowed them to interact with protocol entities and discover more subtypes, thereby enhancing the completeness of the protocol state machine. Pan et al. (2023) addressed the absence of abnormal messages in typically collected offline data by proposing variations such as integer addition and byte-level flipping within normal messages to create abnormal packets. This approach expanded the protocol's input space. However, while these mutation techniques increased sample diversity, they significantly reduced the efficiency of state machine inference.

3 Limitation analysis

Compared to Moore state machines, Mealy state machines offer a more straightforward approach to modeling the interactive behavior of protocols using requests and responses. In a Mealy machine, given the current state and input conditions, the protocol state machine transitions deterministically to a single subsequent state. Therefore, we employ deterministic Mealy state machines to describe the state transitions of protocols. They are defined as follows:

Definition 1 A deterministic Mealy state machine is typically represented as a sextuple $M=(S, S_0, I, O, \delta, \lambda)$, where

1. S denotes a non-empty finite set of states;
2. $S_0 \in S$ signifies the initial state;
3. I represents the input alphabet, a finite and non-empty set, also known as the input space;
4. O signifies the output alphabet, another finite and non-empty set, referred to as the output space;
5. $\delta: S \times I \rightarrow S$ is the state transition function;
6. $\lambda: S \times I \rightarrow O$ is the output function.

In the subsequent text, "Mealy state machine" refers exclusively to deterministic Mealy state machines.

Establishing the input and output alphabets is a critical step in the active inference of protocol state machines. During interactions with protocol entities, both the inputs and outputs are protocol messages. Let the input message be denoted as I^* and the output message as O^*. For example, a protocol message containing an integer field of four bytes can lead to up to 2^{32} scenarios (Székely et al., 2021). However, due to uncertainties such as message length, directly using the various message values as the input and output alphabets would result in an impractically large alphabet. Therefore, it is necessary to classify protocol messages and establish mapping relationships $I^* \rightarrow I$, $O^* \rightarrow O$, abstracting messages into a more manageable input and output alphabet. Existing literature on protocol reverse engineering does not clearly define message types; in this paper, we consider messages of the same type to have the same request/response keywords.

Current research on the active inference of state machines often assumes that a complete protocol input space can be obtained after classifying messages. However, this assumption is idealistic because the message domain constructed from collected offline traffic is likely to be incomplete.

1. Partial message loss in offline traffic. A normal communication session sequence can be represented as a request–response tag sequence:

$$s=\{i_1, o_1, i_2, o_2, \cdots, i_k, o_k, \cdots | i_k \in I, o_k \in O\}. \quad (1)$$

Here, any message transition in the session is represented by (i_k, o_k), where i_k denotes the request message and o_k denotes the response message. When collecting offline traffic, partial loss of i_k or o_k may occur due to network congestion or improper operation.

2. Lack of subtype samples in offline traffic. The same type of request may have different response types; for example, different parameters in SETUP messages of the RTSP protocol can lead to different output types. The collected offline traffic rarely ensures full coverage of subtypes, leading to the loss of some states or state transitions.

3. Lack of samples of certain types in offline traffic. Different types of messages generally correspond to different actions interacting with the protocol entity, and we cannot guarantee that the collected traffic covers all actions specified by the protocol.

Various approaches have been proposed in response to these issues. Sun et al. (2022) proposed a state tag sequence compensation algorithm that logically fills the communication sequence. Székely et al. (2021) extracted subtypes based on existing packets but did not expand abnormal subtypes. Pan et al. (2023) added a mutation module to explore abnormal subtypes, but their method of sequential mutation of all bytes significantly increased the cost.

4 Method

In response to the aforementioned challenges, this paper presents an active inference method for protocol state machines based on incomplete message domains called Act_Infer. Building on previous research, this method is rooted in the MAT framework. Fig. 2 depicts the flowchart of the Act_Infer method, which primarily consists of five main components, preprocessing, mapper construction, traffic deduplication, active inference, and counterexample generation. Throughout the entire process, the input consists of collected offline traffic. Act_Infer establishes the mapping relationship between the alphabet and the protocol message by analyzing the offline traffic. Then, it proactively constructs the test sequences to engage with the protocol entity and generates a hypothetical protocol state machine M based on interaction information. If a counterexample causes M to fail to meet equivalence judgment, it iterates to generate a new hypothesis. This loop continues until the equivalence condition is satisfied.

4.1 Preprocessing

When it comes to analyzing protocols through tracing, using offline traffic collected from real network environments is incredibly valuable. It can significantly reduce the number of connections and interactions needed with protocol entities during active state machine inference. However, incomplete situations, as described in Section 3, may arise due to factors such as network latency and limited data sources. Therefore, it is important to preprocess the original session traffic to complete missing sessions, improve the utilization of data resources, and expand the input space as much as possible through deterministic mutation.

4.1.1 Session complementation

To deal with the issue of partial message loss in protocol communication sessions $s=\{i, i, i, o, o\}$, Sun et al. (2022) proposed inserting a null character τ to replace request or response message tags, ensuring that $s=\{(i, \tau), (i, \tau), (i, o), (\tau, o)\}$. However, this method only ensures that the communication sequence is logically complete and cannot be applied to active inference tasks. Therefore, we develop an algorithm (supplementary materials, Section 1.1.1) for session response state tag completion that restores the missing message tag sequence through interaction with the target protocol entity.

4.1.2 Deterministic variation

In a reverse analysis of unknown protocols, it is challenging to determine the boundaries of message

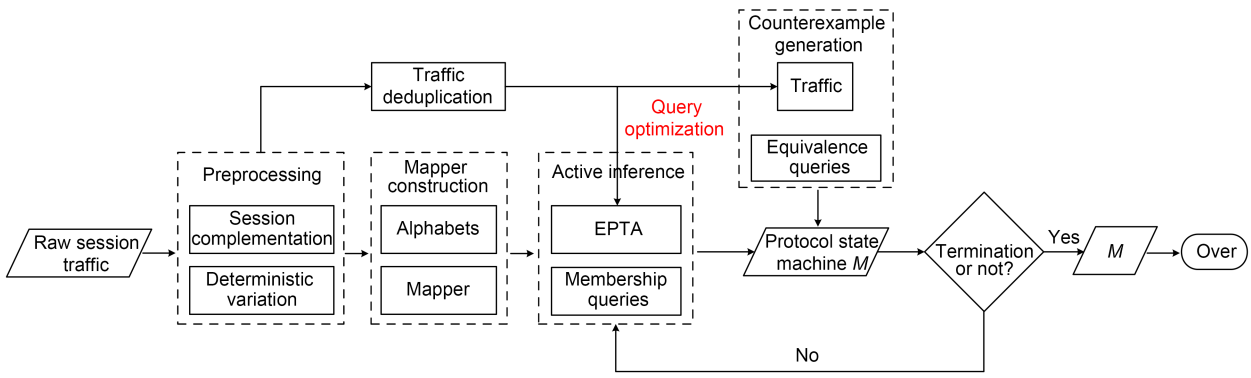

Fig. 2 Act_Infer—active inference process of the protocol state machine based on the incomplete message domain

types and the number of their subtypes. Researchers typically use message types and subtypes derived from existing offline data as the input space for state machines, which limits the exploration of unknown boundaries.

Offline data usually contain only normal traffic. By creating abnormal traffic, it is possible to effectively uncover subtypes within protocols. Starting with the request message sequence seq=$m_1 \cdot m_2 \cdot m_3$, as a seed, each request message is mutated in turn, resulting in the sequence seq=mutate(m_1) $m_2 \cdot m_3$. This new sequence is then input into the SUL to obtain the corresponding output sequence. The output is checked for consistency with existing session sequences. Pan et al. (2023) proposed applying deterministic mutation strategies from American fuzzy lop (AFL), such as byte flipping, integer arithmetic, and dictionary transformations, to mutate messages on a per-byte basis. However, mutating messages on a per-byte basis can lead to numerous invalid mutations. As shown in Table 2, even though the last two messages are abnormal, their response types remain consistent and do not expand the input/output space or state transition relationships of the protocol state machine. Therefore, it is advisable to consider segmenting messages before mutation: for text-based protocols, special symbols can be used as delimiters; for binary protocols, methods such as information entropy or probabilistic inference can be employed for segmentation. Within each message segment, a single byte is randomly selected for mutation.

We devise heuristic rules for analyzing both text-based and binary protocols to understand message types. Text-based protocols often feature identifiable semantic fields related to message types, such as OPTIONS, SETUP, and PLAY, within the RTSP protocol. The protocol's functionality is dedicated to its semantics, and expansion can be accomplished by selecting or associating with similar protocols. Conversely, binary protocols lack obvious semantic information and exclusively differentiate protocol message types using distinct values. While iterating through all possible values may be ideal, in the worst-case scenario, testing 256 different types would be necessary when bytes are taken as the unit, making protocols with dozens of message types relatively complex. Using random values to represent different message types is generally impractical for protocol designers, as it would add complexity to protocol usage. Consequently, fields related to message types tend to exhibit regularity, as shown in Table 3, where the command numbers of the SMB2 protocol consist of consecutive values within the range of 0x0000 to 0x0012. When confronted with an unknown binary protocol, the regularity of known values obtained during protocol format inference can be summarized. Subsequently, values with high confidence are selected for testing. Once the values related to message types are determined, compliant messages can be constructed based on the protocol specifications using offline data and sent to the SUL. Subsequently, protocol message types can be expanded based on the response.

Table 2 RTSP protocol part of the SETUP type message results in byte variations

Type	Protocol message	Response code
SETUP	rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1	200
SETUP	xtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1	404
SETUP	rxsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1	404

Table 3 SMB2 mapping table of protocol command numbers

Protocol command	Value	Protocol command	Value	Protocol command	Value
NEGOTIATE	0x0000	SESSION_SETUP	0x0001	LOGOFF	0x0002
TREE_CONNECT	0x0003	TREE_DISCONNECT	0x0004	CREATE	0x0005
CLOSE	0x0006	FLUSH	0x0007	READ	0x0008
WRITE	0x0009	LOCK	0x000A	IOCTL	0x000B
CANCEL	0x000C	ECHO	0x000D	QUERY_DIRECTORY	0x000E
CHANGE_NOTIFY	0x000F	QUERY_INFO	0x0010	SET_INFO	0x0011
OPLOCK_BREAK	0x0012				

4.2 Mapper construction

The mapper plays a crucial role in translating protocol messages into input/output alphabets. It consists of two key components: the abstraction module and the concretization module. The abstraction module is responsible for converting $I^*/O^* \rightarrow I/O$, enabling the extraction of protocol types from messages for building the state machine. The concretization module functions as the inverse of the extraction type, crafting request messages based on query sequences in accordance with protocol specifications and transmitting them to the SUL.

In practical implementation, the concretization process involves selecting request messages from offline data based on the mapping relationship and sending them to the SUL. However, certain protocols assign random values to specific fields to uniquely identify the current session. Fang et al. (2021) categorized these fields as follows.

1. Identity document (ID): increments to identify the order of messages within a session.

2. Session: generates a unique identifier for the current session, which must be used throughout subsequent interactions until the session ends.

3. Time stamp: indicates the time at which the message was generated.

4. Challenge-response: In this field, one party generates a challenge with a random number, and the other party responds based on the challenge; if there is a mismatch, the session will be terminated.

It is essential to make specific adjustments to these fields before replaying messages. Otherwise, the protocol entity will not reach the expected state, and it will be impossible to deduce a well-structured protocol state machine. For example, considering the RTSP protocol and its Live555 implementation, Table 4 displays part of the alphabet and mapping relationships, and the upper part of Fig. 3 illustrates the state machine inferred by directly replaying messages, while the lower part of Fig. 3 shows the state machine obtained after adjusting the ID and session fields.

4.3 Traffic deduplication

Offline data often contain a significant amount of traffic generated by repetitive operations. For passive inference methods, this statistical information can be useful in revealing the distribution patterns of protocol interactions. However, for active inference, repeated or similar session sequences do not contribute to discovering new input/output types or state transition relationships. Removing these redundant sequences can effectively reduce the data volume, facilitating the subsequent construction of the extended prefix tree and counterexample queries.

Definition 2 Given $a,b \in I$, the connection between a and b is denoted as $a \cdot b$, where the symbol $\cdot$ can be omitted. A finite sequence composed of elements from I is called an input sequence, denoted as w_I, whose length is denoted as $|w_I|$. The set of these sequences is denoted as Σ_I. Similarly, the set of output sequences is denoted as Σ_O.

Definition 3 Given w_I, $v_I \in \Sigma_I$, if $\exists u_I \in \Sigma_I$ such that $w_I = v_I \cdot u_I$, then v_I is referred to as a prefix of w_I, denoted as $v_I \sqsubseteq_{\text{pref}} w_I$.

For protocols, when the input messages are consistent in the same state, the output messages from the protocol entity are uniquely determined. Therefore, during deduplicating traffic, it is sufficient to consider only the input message sequences. For any two i/o interaction sequences seq_i and seq_j, we extract the output labels i_k from (i_k, o_k) to form the input sequences w_{I_i} and w_{I_j}. The deduplication rule is as follows: if $w_{I_i} \sqsubseteq_{\text{pref}} w_{I_j}$, then retain seq_j; if $w_{I_j} \sqsubseteq_{\text{pref}} w_{I_i}$, then retain seq_i; if neither condition is satisfied, then seq_i and

Table 4 RTSP alphabet and protocol message mapping (part)

Alphabet	Protocol message
OPTIONS	OPTIONS rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest RTSP/1.0
DESCRIBE	DESCRIBE rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest RTSP/1.0
SETUP01	SETUP rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0
SETUP02	SETUP rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0 Session: 000022B8
PLAY	PLAY rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/ RTSP/1.0
TEARDOWN	TEARDOWN rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/ RTSP/1.0

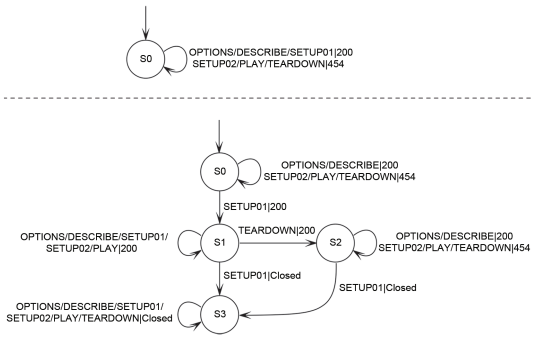

Fig. 3 RTSP protocol state machine obtained using different replay modes

seq_j are not considered duplicate or similar sequences. Based on this rule, we design a traffic deduplication algorithm (supplementary materials, Section 1.2).

4.4 Active inference

Active inference of protocol state machines constitutes the nucleus of the entire process. In alignment with previous research efforts (Wang C et al., 2015; Székely et al., 2021; Pan et al., 2023), this paper uses the L_{M}^{+} algorithm. However, we augment its capabilities by developing an EPTA and formulating a response-based query optimization algorithm. These enhancements aim to bolster the efficiency of inference procedures.

4.4.1 EPTA

The EPTA capitalizes on the wealth of session information embedded within offline data, optimizing the efficiency of membership and equivalence queries by curbing the need for excessive interactions with the SUL. Leveraging the structure of the prefix tree, which is derived from N-ary tree concepts and a variant of the hash tree, facilitates swift and effective searches.

Based on the prefix tree structure and the deduplication session tag sequence set $seqs_{\mathrm{TD}}$, we construct an EPTA, as shown in Fig. 4. The edges represent input labels i_k from (i_k, o_k) pairs, and nodes correspond to output labels o_k (supplementary materials, Section 1.3.1).

4.4.2 Response-based query optimization algorithm

To enhance the efficiency of protocol interaction queries, we propose two optimization rules based on the characteristics of protocol communication.

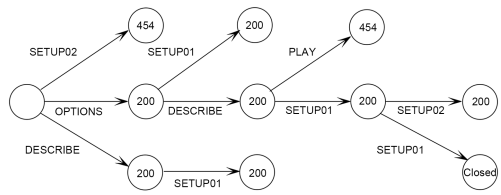

Fig. 4 RTSP protocol extended prefix tree (part)

1. Early termination on connection closure: If a query message sent to the SUL results in the closure of the connection, the remaining part of this query can be terminated. The output sequence will then be extended with a 'Closed' status to match the length of the remaining unprocessed input sequence.

2. Prefix-based optimization: Let w_I be the input sequence leading to connection closure and let w_{closed} be the shortest such sequence. For any future input sequence w_{I_i}, if w_{closed} is a prefix of w_{I_i}, the interaction with the SUL can be skipped. The response can be directly derived from the EPTA.

These rules form the basis for the response-based query optimization algorithm (supplementary materials, Section 1.3.2).

4.5 Counterexample generation

Based on the results of active inference, we can hypothesize a protocol state machine M_{M}. To validate this candidate state machine, an equivalence approximation assessment is necessary. Initially, offline data can be used for this equivalence determination to check whether M_{M} can handle normal data flows. However, relying solely on normal data access is insufficient to ensure the completeness of M_{M}. Thus, generating test samples for equivalence queries is crucial.

In the original L_{M}^{+} algorithm, test samples are randomly generated from a uniformly distributed input sequence based on I. This replay traffic is then sent to the SUL, and the output is compared with the output from M_{M}. Any discrepancies found are used as counterexamples. Angluin (1987) described the probabilistic relationship between the number of test samples and the likelihood that M_{M} is the actual state machine, expressed as

$$N=\left(\log\left(1/\gamma\right)+\log 2\cdot\left(n+1\right)\right)/\beta, \tag{2}$$

where N represents the number of test samples, γ is the confidence level, n represents the number of

candidate state machines M_M that have been generated, and β represents the generalization error. If no counterexamples are found after N tests, it can be considered that the probability that M_M is a β-approximation of the actual state machine is at least $1-\gamma$.

Although Eq. (2) provides a termination condition for equivalence determination, randomly constructing test samples has its limitations.

1. State coverage: The purpose of equivalence queries is to find missing states in the hypothesized state machine M_M. Test sequences should be designed to explore new states by covering the existing states in M_M.

2. Sample reuse: If a test sample is not a counterexample for the first $n-1$ candidate state machines, it cannot be a counterexample for the nth candidate state machine. Therefore, sequences used in previous equivalence determinations should be avoided.

To address these limitations, we propose a state transition-based random counterexample generation method, leveraging the structure of the observation table (OT) maintained in the L_M^+ algorithm (the right part of Fig. 5 at steps ① and ②).

The OT records the results of input sequence queries and is defined as follows:

Definition 4 The OT is a triplet (S_M, E_M, T_M), where S_M is a prefix-closed set based on I, E_M is a suffix-closed set based on I, and T_M maps $(S_M \cup S_M \cdot I) \times E_M$ to O. For $\forall s \in S_M \cup S_M \cdot I$ and $e \in E_M$, $T_M(s \cdot e)=o$, where $o \in O$.

The OT must satisfy the following properties:

Definition 5 Consistency: $\forall s, t \in S_M \cup S_M \cdot I$, if $\forall e \in E_M$, $T_M(s \cdot e)=T_M(t \cdot e)$, then s and t are consistent, denoted $s \cong_{E_M} t$, and $[s]$ represents the row equivalent to s.

Definition 6 Closure: $\forall t \in S_M \cup S_M \cdot I$, $\exists s \in S_M$ such that $s \cong_{E_M} t$.

It is assumed that the hypothesized state machine is shown in the left part of Fig. 5. S_M and E_M are both non-empty sets, and the OT is initialized with $S_M=\{\epsilon\}$, $E_M=I$. As shown in the right part of Fig. 5 at step ①, each character in the input alphabet forms a column, and each row represents $s \in S_M \cup S_M \cdot I$. It is evident from step ① that the table does not satisfy closure; therefore, rows where characters $a \in S_M \cdot I$ are added to S_M, expanding the OT until it is both consistent and closed (the right part of Fig. 5 at step ②). Once the OT is consistent and closed, a hypothesized state machine M_M can be constructed as follows:

Definition 7 $M_M=(S_M, S_{M0}, I, O, \delta_M, \lambda_M)$, where

1. $S_M=\{[s] \mid s \in S_M\}$;
2. $S_{M0}=[\epsilon]$;
3. $\delta_M([s], i)=[s, i]$, $\forall s \in S_M$, $i \in I$;
4. $\lambda_M([s], i)=T_M(s, i)$, $\forall i \in I$.

Therefore, based on the OT shown in the right part of Fig. 5 at step ②, the state machine depicted in the right part of Fig. 5 at step ③ can be obtained.

Using the OT, Shahbaz (2008) proposed a counterexample separation algorithm that retains only inconsistent rows in S_M, ensuring that the number of elements in S_M matches the number of states in the hypothesized protocol state machine. This approach facilitates the exploration of missing states by performing mutation operations (insertion, deletion, and replacement) on the input sequences in S_M, narrowing the scope of random test sample construction and improving the efficiency of counterexample queries.

Fujiwara et al. (1991) adopted a similar approach in their Wp equivalence query optimization algorithm, constructing equivalence query sequences based on a closed and consistent OT as $S_M \cdot I^{n_a-n_c} \cdot T_M$, where n_a is the actual number of states and n_c is the number of states in the current hypothesized state machine. For

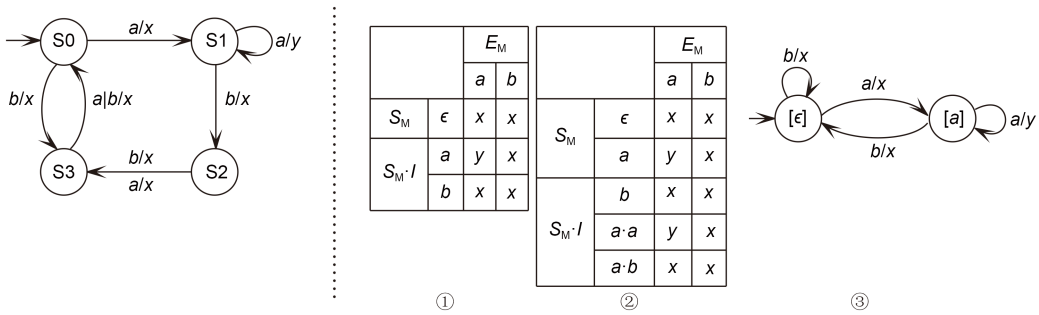

Fig. 5 Observation table creation and update (part)

unknown protocols, n_a is unknown, so the sequence can be simplified to $S_M \cdot I^{\alpha} \cdot T_M$, where α represents the desired exploration depth. While the Wp method can increase accuracy, it requires numerous connections and interactions with the SUL, which is less efficient for analyzing unknown complex systems. A detailed comparative analysis of the execution efficiency of the random and Wp algorithms is provided in Section 5.3.

5 Evaluation and analysis

Due to the absence of character encoding and clear delimiters in binary-based protocols, it is often considered a separate research topic from text-based protocols when evaluating the feasibility of protocol format inference methods. However, when inferring about protocol state machines, our focus is on the interactions between protocol entities. Therefore, there is no need to treat these two types of protocols differently in terms of state machine inference methods. In this section, we select two simple text-based protocols to evaluate and analyze the proposed active inference method for protocol state machines based on incomplete message domains. The assessment involves interactions with the implementation entities of the RTSP and SMTP protocols to evaluate the completeness of the inferred protocol state machines. For better experimental evaluation, we assume that we have obtained a complete and accurate protocol format and message type to create the state machine.

5.1 Evaluation setup

5.1.1 Experimental environment

The experimental process is conducted on a virtual machine with 8 GB of memory operating on Ubuntu 18.04. Python 3.9 serves as the programming language, and the implementation is built upon the AALpy open-source project (AALpy, https://github.com/DES-Lab/AALpy) and its associated environment. AALpy, developed by Muškardin et al. (2022), is a lightweight automaton learning library written in Python. It provides a range of active learning algorithms and equivalence query algorithms, including L^*, L^*_M, L^+_M, and Wp.

5.1.2 Experimental objects

This study focuses on analyzing two protocols and their respective implementations.

1. SMTP protocol and Exim implementation (Exim): Exim, developed by the University of Cambridge, is a message transfer agent (MTA) used to connect to the Internet on Unix systems. The versions analyzed in this study are Exim 4.97 and 4.93.

2. RTSP protocol and Live555 implementation (Live555): Live555 is an open-source C++ project that provides solutions for streaming media. It supports standard streaming media transmission protocols and various audio and video encoding formats. The versions examined are Live555 20240228 and 20200809.

5.1.3 Experimental data

The dataset used in this study comprises two primary sources: a public dataset from the Universitat Politècnica de Catalunya (UPC) (Bujlow et al., 2015) and a locally collected dataset. Table 5 presents a detailed breakdown of message types extracted from these datasets.

Table 5 Test datasets

Protocol	Source	Number of packages	Number of sessions	Size (MB)
SMTP	UPC_smtp	194 572	67	55.3
	LOCAL_smtp	51 780	152	20.14
RTSP	LOCAL_ac3	1643	10	0.79
	LOCAL_mp3	6984	10	2.31
	LOCAL_mpg	16 538	10	6.56
	LOCAL_wav	9869	10	3.47
	LOCAL_webm	15 448	10	5.69
	LOCAL_aac	5796	10	1.98
	LOCAL_mkv	10 468	10	4.08

5.2 Protocol state machine active inference results

In this subsection, Act_Infer relies on the optimized random counterexample query algorithm to perform repeated experiments on multiple versions of the RTSP and SMTP protocols.

The inference results for the RTSP protocol and its implementation Live555 are shown in Fig. 6. Here, Figs. 6a and 6b correspond to versions 20200809 and

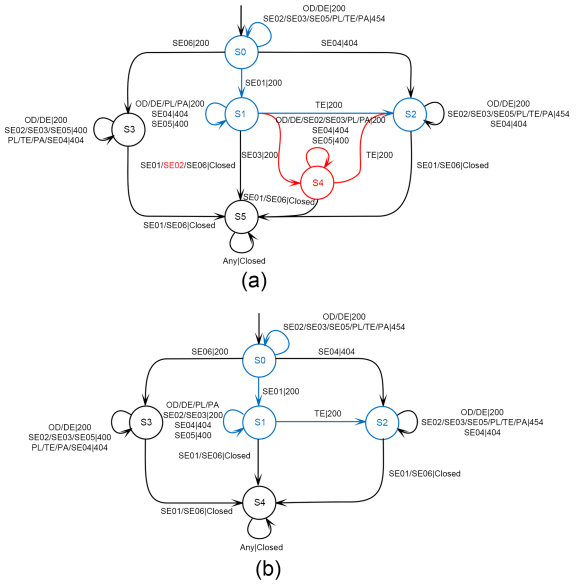

Fig. 6 Act_Infer inferred Live555 protocol state machine: (a) Live555 20200809; (b) Live555 20240228. OD: OPTIONS; PL: PLAY; DE: DESCRIBE; TE: TEARDOWN; SE: SETUP; PA: PAUSE. References to color refer to the online version of this figure

20240228, respectively. Due to the simplicity of the RTSP protocol, the resulting state space for Live555 is relatively small, with straightforward state transition relationships. According to the RFC documentation, the blue sections in Fig. 6 represent state transitions executed according to normal logic, following the basic session sequence {(OPTIONS, 200), (DESCRIBE, 200), (SETUP, 200), (PLAY, 200), (TEARDOWN, 200)}. It is important to note that the RTSP protocol analysis is based on sessions. Therefore, if the session value changes, even if the protocol implementation can continue normally, the communication round is considered ended and marked as "Closed."

Based on the deterministic mutation method described in Section 4.1, various subtypes of the SETUP message type are added to the offline data message types, as distinguished by numerical labels in Fig. 6. Additionally, considering the nature of streaming media protocols and integrating the semantics of existing message types, a PAUSE message type is constructed. The detailed mapping relationships of the SETUP subtypes to the messages are shown in Table 6. Specifically, SETUP01, SETUP03, and SETUP06 correspond to messages existing in the offline data. The differences lie in SETUP01 and SETUP03 selecting different tracks when executing the same streaming media file, while SETUP01 and SETUP06 execute different types of streaming media files. The remaining protocol messages are new messages obtained by mutating SETUP01 or SETUP03, with the mutation points indicated in red.

Comparing the state machines obtained from the two versions of Live555 in Fig. 6, it can be observed that the older version, 20200809, has one additional state compared to the newer version, 20240228. The main reason for this difference is that sending two consecutive SETUP commands to the same track leads to a use-after-free vulnerability and the crash of the daemon, causing Live555 to actively close the connection. This vulnerability, identified by the number CVE-2021-38381 (https://nvd.nist.gov/vuln/detail/CVE-2021-38381), is fixed in versions after 20210809.

Figs. 7 and 8 depict the state machines inferred from the SMTP protocol and its implementation Exim. Fig. 7 corresponds to the inference results for version 4.93, while Fig. 8 corresponds to version 4.97. According to the RFC documentation, the basic logic of the SMTP protocol is relatively straightforward; the blue sections in the diagrams represent the state transitions executed according to normal protocol logic. These transitions correspond to the fundamental session sequence: {(EHLO, 250), (MAIL, 250), (RCPT, 200), (DATA, 354), (MSG, 250), (QUIT, 221)}. Upon

Table 6 Mapping between SETUP subtypes and packets

Alphabet	Protocol message
SETUP01	SETUP rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0
SETUP02	SETUP rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0 Session
SETUP03	SETUP rtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track2 RTSP/1.0 Session
SETUP04	SETUP xtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0
SETUP05	SETUP xtsp://127.0.0.1:8554/mpeg1or2AudioVideoTest/track1 RTSP/1.0 Session
SETUP06	SETUP rtsp://127.0.0.1:8554/mp3AudioTest/track1 RTSP/1.0

References to color refer to the online version of this figure

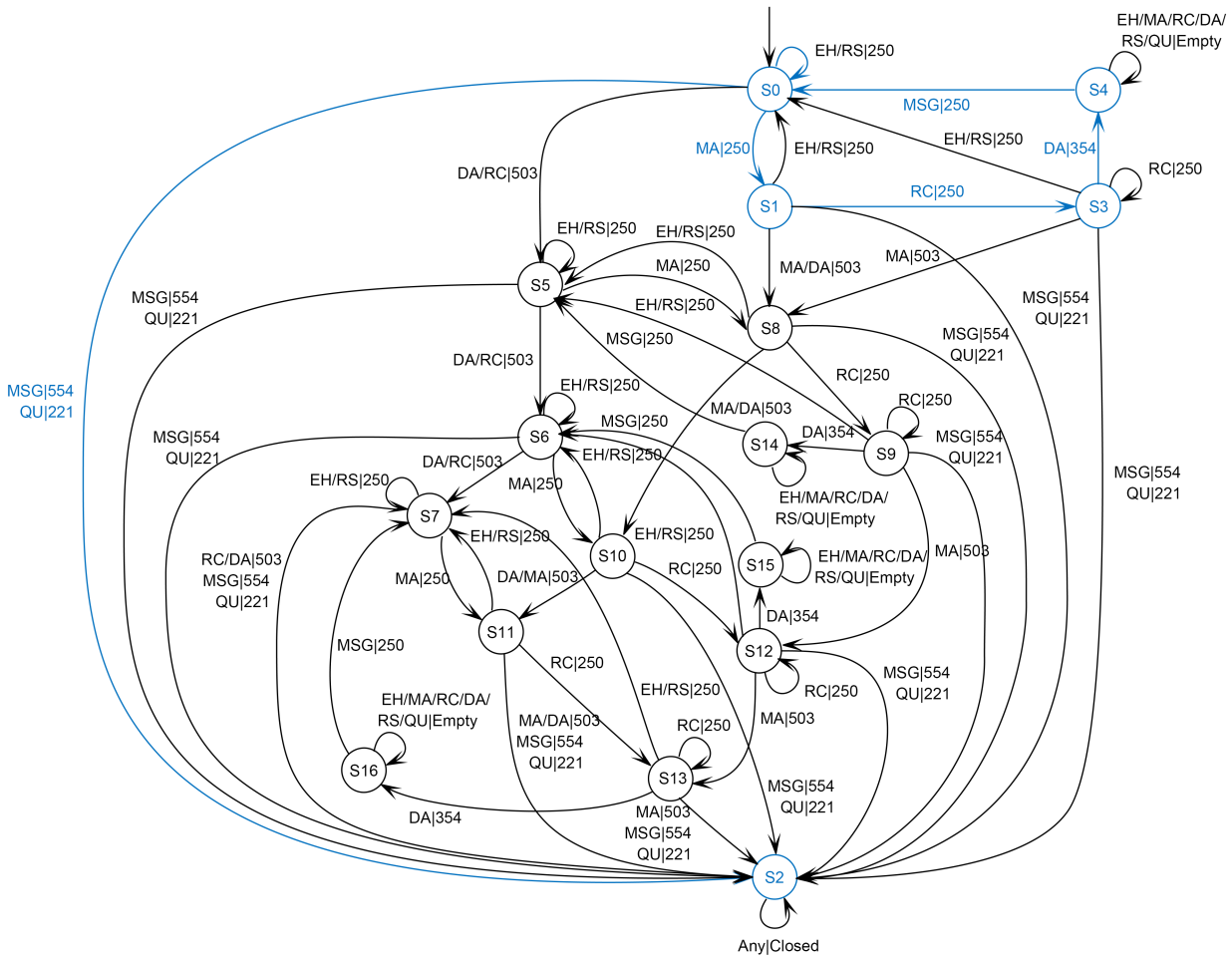

Fig. 7 Inferred protocol state machine of Exim 4.93 generated by Act_Infer. EH: EHLO; MA: MAIL; RC: RCPT; DA: DATA; RS: RSET; QU: QUIT. References to color refer to the online version of this figure

the completion of this sequence, if the QUIT command is not executed in Exim 4.93, it results in a cyclic transition between states S0→S1→S3→S4→S0. In contrast, in Exim 4.97, it forms a cyclic transition between states S30→S34→S35→S8→S30.

Although the SMTP protocol has a straightforward normal logic flow, similar to the RTSP protocol, the state machine inferred from Exim is notably more complex than that of Live555. This increased complexity is due to Exim's detailed exception-handling mechanisms, which are employed alongside the protocol's basic mail transfer functionalities. For instance, in Exim 4.93, the QUIT command terminates the connection, and if the MSG mail information is not sent correctly, the server will return an error code "554" and close the connection. Moreover, if an abnormal sequence of request messages leads the server to return four or more "503" error codes, the connection will also be closed. An example of such a sequence is "RCPT-DATA-RCPT-DATA-MAIL," which results in the response sequence "503-503-503-503-Closed." The Exim documentation (Exim master documentation, https://www.exim.org/exim-html-current/doc/html/spec_html/index.html) details this exception handling in the default configurations for the parameters "smtp_max_synprot_errors" and "smtp_max_unknown_commands."

Comparing the state machines of Exim 4.93 and Exim 4.97 reveals significant differences in complexity and adherence to RFC specifications. The state machine for Exim 4.97 is twice as large as that in Exim 4.93 and exhibits more intricate state transitions. One key difference is that Exim 4.93 does not strictly enforce the RFC-mandated requirement that mail processing must start with the EHLO command. In contrast, Exim 4.97 (Exim master documentation) addresses

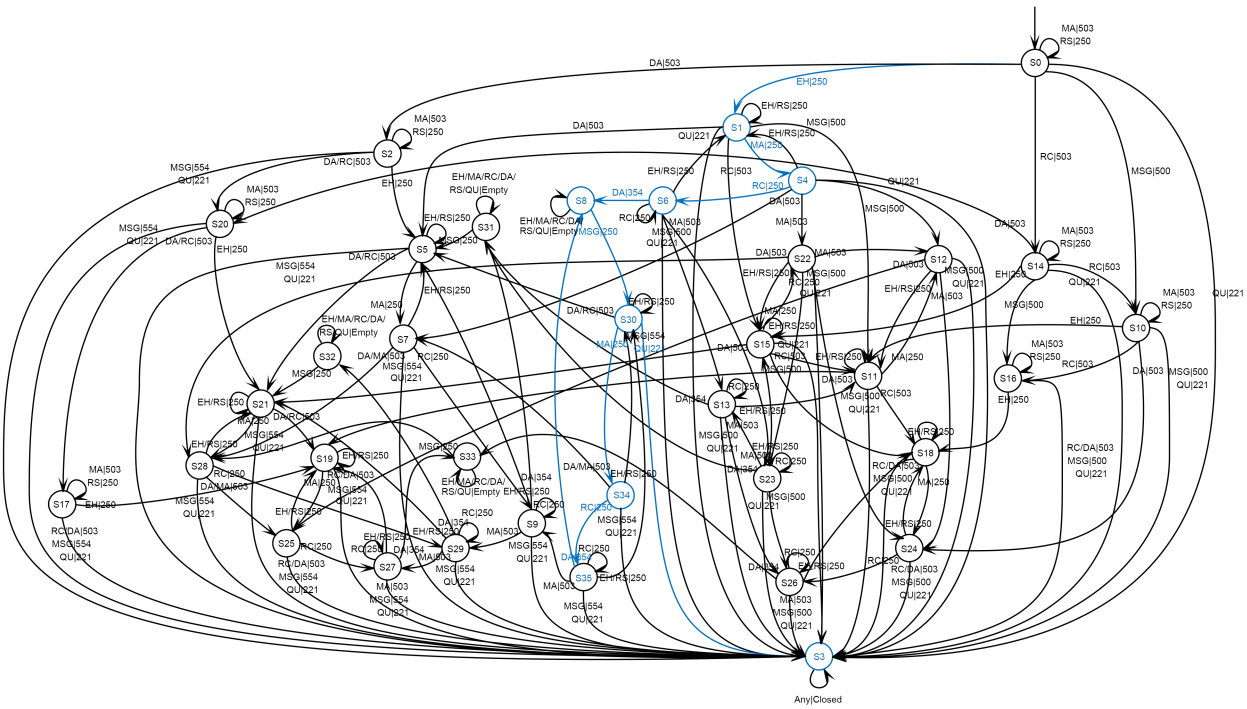

Fig. 8 Inferred protocol state machine of Exim 4.97 generated by Act_Infer. References to color refer to the online version of this figure

this with the parameters "helo_verify_hosts" and "hosts_require_helo," ensuring compliance. Additionally, in handling abnormal request sequences, Exim 4.93 closes the connection after four consecutive "503" error codes, such as with the sequence "MAIL-MAIL-MAIL-MAIL-MAIL" without an initial EHLO. However, Exim 4.97 continues to return "503" without terminating the connection, allowing for uninterrupted server operation despite repeated errors. These enhancements in Exim 4.97 result in a more complex but robust state machine, reflecting better compliance and improved error-handling mechanisms.

5.3 Efficiency analysis

This subsection aims to delve into the execution efficiency of the methods proposed in this paper and validate the effectiveness of the optimization techniques discussed in Sections 4.4 and 4.5. Our evaluation focuses on the following four key aspects: execution time, number of queries, number of connections, and number of interactions. Given that active inference-based methods typically comprise two phases—the state machine learning and construction phase involving membership queries and the equivalence determination phase involving equivalence queries—we tailor our evaluation metrics to differentiate between these phases for a more comprehensive assessment. The specifics of these evaluation metrics and their descriptions are outlined in Table 7.

In our learning experiments, we used Live555 20240228 and Exim 4.97 as our subjects, benchmarking against the original algorithms implemented in AALpy. Throughout the state machine learning and construction phase, we employed the original L_{M}^{+} algorithm as a baseline. For the equivalence determination phase, we compared two approaches: the random counterexample generation algorithm and the Wp equivalence query optimization algorithm. Notably, certain parameters required presetting during the equivalence determination phase. For the random counterexample generation algorithm, we assumed a confidence level γ of 0.01 and a generalization error β of 0.01, as per Eq. (2). For the Wp equivalence query optimization algorithm, we set n_{a}–n_{c} to 1, indicating an exploration depth α of 1 in the current hypothesis state.

Fig. 9 presents a detailed comparative analysis of execution efficiency. The blue and gray columns denote L_{M}^{+} (random) and L_{M}^{+} (Wp) implemented by AALpy, while the red and purple columns represent the optimized Act_Infer (random) and Act_Infer (Wp),

Table 7 Evaluation indexes and descriptions

Evaluation metric	Description	Phase	Indicator
Execution time	Time consumed for program execution	Learning	T_l
		Equivalence	T_e
		All	T_a
Number of queries	Number of query sequences during state machine inference	Learning	Q_l
		Equivalence	Q_e
		All	Q_a
Number of connections	Number of Socket connections established with SUL	Learning	C_l
		Equivalence	C_e
		All	C_a
Number of interactions	Number of request messages sent to the SUL	Learning	IL
		Equivalence	IE
		All	IA

respectively. It is evident from the figure that Act_Infer, in comparison with the original L_M^+, requires nearly the same number of queries as SUL, as the optimized methods retain the fundamental decision criteria. Additionally, during both the state machine learning and construction phase and the equivalence determination phase, the required execution time, as well as the actual number of connections and interactions with the SUL, is notably reduced.

Table 8 presents an exhaustive evaluation of various metrics across the learning experiments conducted on Live555 20240228 and Exim 4.97. The evaluation metrics encompass total execution time (T_a), total number of queries (Q_a), total number of connections (C_a), and total number of interactions (IA) involved in the inference process. Act_Infer, using both random and Wp optimization approaches, demonstrates substantial improvements over the traditional L_M^+ algorithm implemented by AALpy. Notably, Act_Infer achieves an average reduction of approximately 40.7% in execution time compared to L_M^+, showcasing enhanced efficiency. Furthermore, Act_Infer significantly reduces the number of connections and interactions by approximately 28.6% and 46.6%, respectively, contributing to a more streamlined inference process.

Considering that the exploration depth α remains constant, the Wp algorithm meticulously scrutinizes the accuracy of the protocol state machine through exhaustive traversal. Conversely, the random algorithm's generation of counterexamples, given the fixed confidence level γ and generalization error β, is primarily determined by the number of candidate state machines. A comparative analysis of the two equivalence query algorithms underscores the notable efficiency advantage of the random counterexample generation method, which requires considerably less execution time and fewer queries to the SUL compared to the Wp algorithm. This discrepancy in execution efficiency becomes more pronounced with an increase in the number of states in the candidate state machine and a larger input space. For instance, during the inference of the Live555 state machine, the random algorithm's execution time is approximately 23.5% of that of the Wp algorithm, and this difference further decreases to approximately 11.1% when inferring the more intricate Exim state machine. Furthermore, when inferring the state machines of unknown protocols and their implementations, setting α to 1 in the Wp algorithm often proves inadequate. This limitation arises from the insufficient depth of exploration, which may fail to uncover deeper counterexamples. For example, during the inference of Exim 4.93, neither α=1 nor α=2 could unveil the counterexample "RCPT-DATA-RCPT-DATA-MAIL," as detailed in Section 5.2.

Table 9 presents a comparison of the inference results obtained by setting different α values when targeting Exim 4.93. Notably, when α is set to 1 or 2, the inferred results consist of only five distinct states. It is not until α reaches 3 that the number of states in the state machine changes. Furthermore, as α increases, the inference process consumption exhibits an exponential growth trend; for instance, the execution time at α=3 is approximately 37 times that at α=1. Since the counterexample query sequence generated by the random algorithm is uncertain, the inference efficiency of this algorithm is also uncertain. Fig. 10

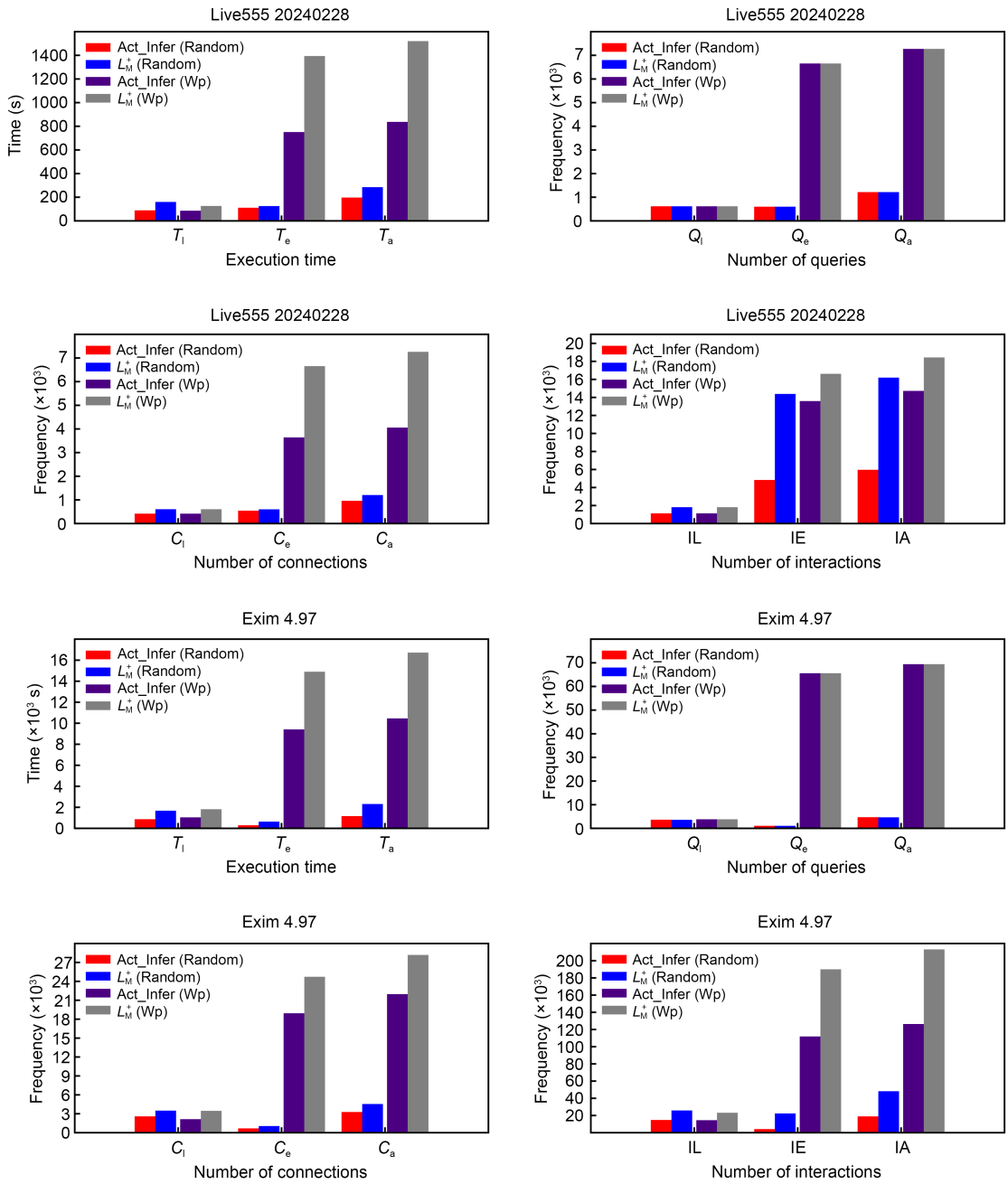

Fig. 9 Comparative analysis of execution efficiency. References to color refer to the online version of this figure

Table 8 Evaluation of various metrics across the learning experiments

Method (Equivalence query algorithm)	Live555 20240228				Exim 4.97			
	T_a (s)	Q_a	C_a	IA	T_a (s)	Q_a	C_a	IA
Act_Infer (random)	197	1216	960	5964	1165	4718	3278	18 947
L_M^+ (random)	285	1216	1205	16 999	2314	4650	4547	48 214
Act_Infer (Wp)	837	7271	4057	14 724	10 458	69 421	21 974	126 377
L_M^+ (Wp)	1520	7271	7260	18 435	16 721	69 421	28 182	213 129

Table 9 Comparison of Wp algorithm inference results for Exim 4.93

Parameter	Number of states	T_a (s)	Q_a	C_a	IA
α=1	5	964	1976	1870	7125
α=2	5	4091	12 257	7659	29 053
α=3	15	35 927	259 548	76 249	297 185

depicts the average efficiency of executing the random algorithm 10 times. The horizontal axis represents the number of states contained in the candidate state machine being inferred, while the vertical axis represents the execution time, number of queries, number of connections, or number of interactions. By recording the consumption required to generate each round of candidate state machines, a line chart is obtained. The red dashed line indicates the average execution efficiency over multiple experiments. To achieve the same inference results, the consumption of the random algorithm is approximately 1/40 that of the Wp algorithm. However, due to the randomness of generating counterexample query sequences, the number of rounds needed to deduce the final state machine in multiple experiments may vary significantly, potentially leading to incomplete state machines being inferred.

When dealing with unknown protocols and their implementations, the Wp algorithm can ensure the uniqueness and certainty of the results once the exploration depth α is determined. However, estimating the complexity of the target state space is challenging. Setting α too low may result in incomplete inferences due to insufficient depth, while setting it too high can exponentially increase the inference time. Consequently, determining the appropriate value for α is extremely difficult. In contrast, the random algorithm offers flexibility by adjusting the length of the equivalence query sequences, which allows it to discover state transitions that require deeper interaction. Furthermore, since each inference with the random algorithm is relatively quick, multiple inferences can be conducted to cover any omitted states, enhancing the overall analysis. Therefore, the random algorithm is more suitable for the rapid analysis of unknown systems due to its adaptability and efficiency in uncovering complex state transitions.

5.4 Comparison analysis

This subsection focuses on Live555 20200809 and Exim 4.93 as the subjects of study. We compare and analyze the state machines inferred using the methods proposed by Sun et al. (2022) and Pan et al. (2023). Fig. 11 illustrates the inference results. Specifically, Figs. 11a and 11c represent the protocol state machines obtained by analyzing the collected offline data using the Sptia-PL method proposed by Sun et al. (2022),

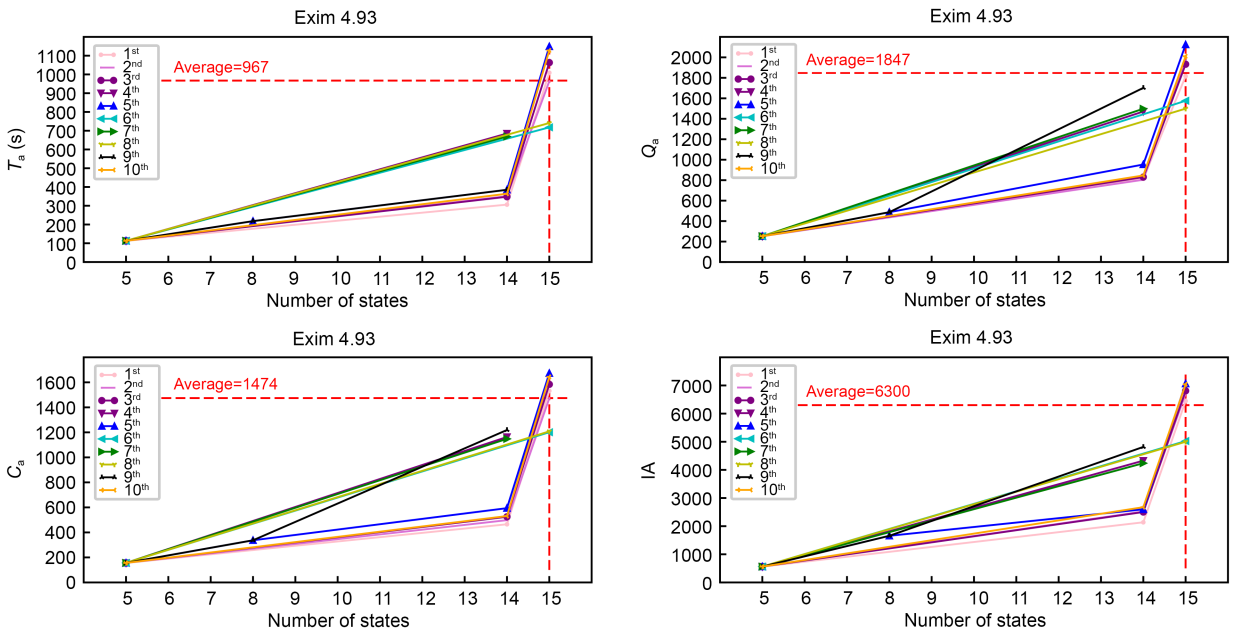

Fig. 10 Average efficiency of 10 random algorithm executions using Exim 4.93 as the object. References to color refer to the online version of this figure

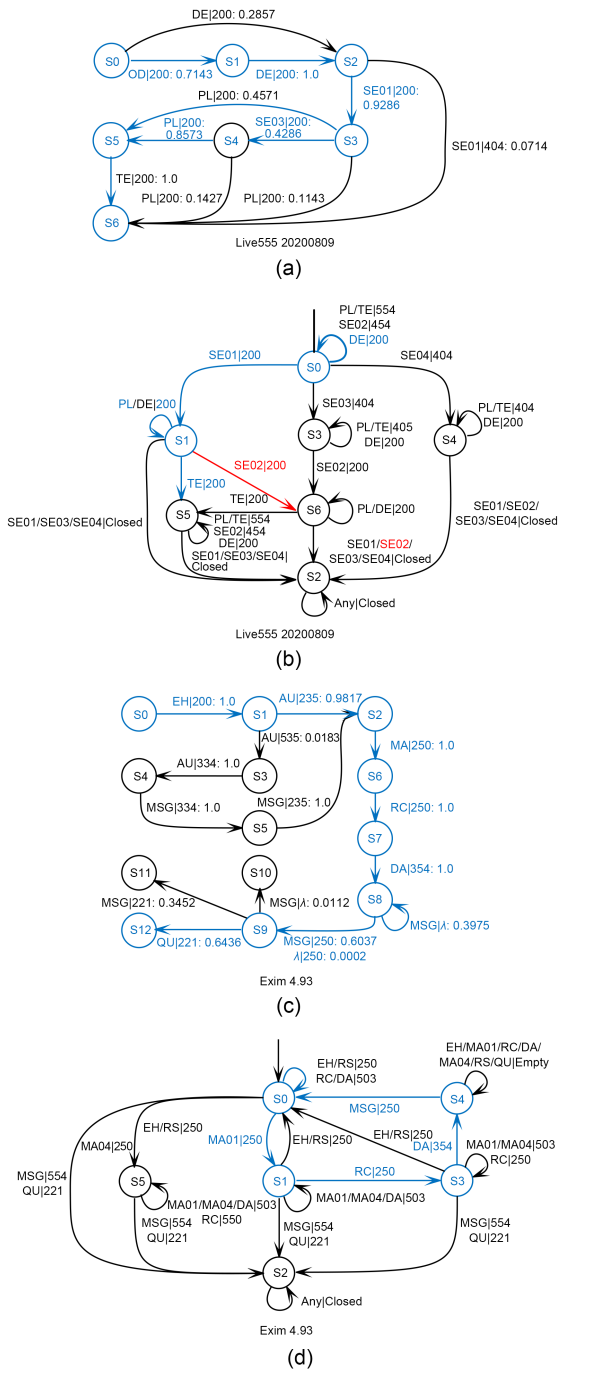

Fig. 11 The protocol state machines inferred using the methods proposed by Sun et al. (2022) (a, c) and Pan et al. (2023) (b, d). AU: AUTH. References to color refer to the online version of this figure

while Figs. 11b and 11d are the state machines derived from the progressive active inference approach employed by Pan et al. (2023). In line with the previous text, the blue sections in Fig. 11 indicate the state transitions executed according to normal logic, whereas the red sections highlight logical flaws present in the implementation of the protocol entities.

When comparing Fig. 11a with Fig. 11b and Fig. 11c with Fig. 11d, it becomes clear that, whether the RTSP or SMTP protocol is used, the normal protocol execution logic can be quickly deduced from offline data. This results in a relatively simple protocol state machine. Generally, by following the direction of the highest calculated probability of state transitions at each state, the normal interaction sequence can be identified, which helps with rapidly analyzing the protocol's behavior. However, Sptia-PL fails to identify the CVE-2021-38381 vulnerability present in Live555 and the behavior that violates the RFC standards in Exim.

In contrast, the state machine created by Pan et al. (2023) and the results discussed in Section 5.2 illustrate that the outcomes obtained from an active inference method are relatively intricate. The normal protocol execution logic occupies only a small part of the state machine space, while the machine covers numerous state transitions under abnormal conditions. These transitions are capable of detecting existing vulnerabilities and behaviors that violate protocol standards. Both methods can also be extended to protocol message subtypes not present in offline data. However, when analyzing more complex protocols, our method demonstrates a more notable advantage. For instance, when examining the state machine of Exim 4.93, Act_Infer deduces 16 distinct states, whereas the results from Pan et al. (2023) include only six different states. Furthermore, the results fail to recognize that when the server receives an abnormal message and continuously responds with the error code "503," an exception-handling mechanism that actively closes the connection is triggered. Upon analyzing the source code, it becomes apparent that Pan et al. (2023) employed the Wp algorithm in the equivalence determination process, and to consider the inference efficiency, they set α to 1. This led to the failure to detect state transitions that require deep interaction to manifest.

6 Discussions

This section delves into the strengths and potential weaknesses of the proposed approach.

6.1 Advantages

Enhanced inference efficiency: This study significantly enhances the efficiency of proactive protocol state machine inference. It achieves this by building a prefix tree based on offline data, creating a response-based query optimization algorithm, and fine-tuning the method for generating random equivalent query samples.

More comprehensive state machines: By employing a deterministic variation method during the preprocessing phase, the research diversifies the messages in the offline data. This action expands the protocol's input space. Consequently, through proactive interaction with the protocol implementation entity, a more comprehensive protocol state machine is achieved.

6.2 Deficiencies

Proactive inference of complex protocols: While the methods proposed in this study yield promising results when applied to lightweight protocols such as RTSP and SMTP, compared with those of the simpler RTSP protocol, the total link times and total interaction times of the SMTP protocol are increased fourfold, and the total execution time is directly increased tenfold. It can be seen that inference efficiency is the key factor restricting the application of active inference technology to complex protocols. Hence, enhancing the efficiency of proactive inference in such scenarios is crucial for future research.

Exploration of deep cyclic states: The protocol state machines detailed in this paper showcase a variety of cyclic states, such as "EHLO|250." Upon comparing this with the primary documentation of Exim 4.97, it becomes apparent that the designers integrate the parameter "smtp_accept_max_nonmail" to counteract denial of service (DoS) attacks. This parameter ensures that the server promptly disconnects after receiving more than 10 consecutive EHLOs, thereby mitigating potential threats. It is crucial to delve into deep cyclic states while maintaining operational efficiency.

7 Conclusions

In this paper, we investigate the challenge of the reverse inference of state machines for unknown protocols and present a novel active inference model for protocol state machines called Act_Infer, which is based on the MAT framework. Act_Infer addresses the issue of incomplete input space resulting from the absence of offline traffic messages by constructing the input space of the protocol state machine through message session sequence completion and deterministic variation. Moreover, it improves the efficiency of active inference of protocol state machines through traffic deduplication, EPTA construction, response-based query optimization, and random counterexample generation based on state transitions. Our experimental analysis of the RTSP and SMTP protocols and their implementations demonstrates that Act_Infer can produce a more comprehensive protocol state machine in less time and with fewer connections and interactions. This highlights the potential of Act_Infer to significantly enhance the inference of protocol state machines.

Contributors

Maohua GUO and Yuefei ZHU investigated and summarized the literature. Maohua GUO drafted the paper. Maohua GUO and Jinlong FEI revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

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

References

Abdulganiyu OH, Ait Tchakoucht T, Saheed YK, 2023. A systematic literature review for network intrusion detection system (IDS). *Int J Inform Secur*, 22(5):1125-1162. https://doi.org/10.1007/s10207-023-00682-2

Angluin D, 1987. Learning regular sets from queries and counterexamples. *Inform Comput*, 75(2):87-106. https://doi.org/10.1016/0890-5401(87)90052-6

Antonakakis M, April T, Bailey M, et al., 2017. Understanding the Mirai botnet. Proc 26th USENIX Security Symp, p.1093-1110.

Antunes J, Neves N, Verissimo P, 2011. Reverse engineering of protocols from network traces. Proc 18th Working Conf on Reverse Engineering, p.169-178. https://doi.org/10.1109/WCRE.2011.28

Bermudez I, Tongaonkar A, Iliofotou M, et al., 2016. Towards automatic protocol field inference. *Comput Commun*, 84: 40-51. https://doi.org/10.1016/j.comcom.2016.02.015

Bossert G, Guihery F, Hiet G, 2014. Towards automated protocol reverse engineering using semantic information. Proc

9th ACM Symp on Information Computer and Communications Security, p.51-62. https://doi.org/10.1145/2590296.2590346

Bujlow T, Carela-Español V, Barlet-Ros P, 2015. Independent comparison of popular DPI tools for traffic classification. *Comput Netw*, 76:75-89. https://doi.org/10.1016/j.comnet.2014.11.001

Chandler J, 2023. Poster: a Monte Carlo ensemble approach to automatically identifying keywords in binary message formats. Proc Network and Distributed System Security Symp.

Chandler J, Wick A, Fisher K, 2023. BinaryInferno: a semantic-driven approach to field inference for binary message formats. Proc Network and Distributed System Security Symp. https://doi.org/10.14722/ndss.2023.23131

Cho CY, Babi ĆD, Shin ECR, et al., 2010. Inference and analysis of formal models of botnet command and control protocols. Proc 17th ACM Conf on Computer and Communications Security, p.426-439. https://doi.org/10.1145/1866307.1866355

de Carli L, Torres R, Modelo-Howard G, et al., 2017. Botnet protocol inference in the presence of encrypted traffic. Proc IEEE Conf on Computer Communications, p.1-9. https://doi.org/10.1109/INFOCOM.2017.8057064

Fang DL, Song ZW, Guan L, et al., 2021. ICS3Fuzzer: a framework for discovering protocol implementation bugs in ICS supervisory software by fuzzing. Proc 37th Annual Computer Security Applications Conf, p.849-860. https://doi.org/10.1145/3485832.3488028

Fujiwara S, Bochmann GV, Khendek F, et al., 1991. Test selection based on finite state models. *IEEE Trans Softw Eng*, 17(6):591-603. https://doi.org/10.1109/32.87284

Gold, EM, 1967. Language identification in the limit. *Inform Contr*, 10(5):447-474. https://doi.org/10.1016/S0019-9958(67)91165-5

Huang YY, Shu H, Kang F, et al., 2022. Protocol reverse-engineering methods and tools: a survey. *Comput Commun*, 182:238-254. https://doi.org/10.1016/j.comcom.2021.11.009

Kleber S, Kopp H, Kargl F, 2018. NEMESYS: network message syntax reverse engineering by analysis of the intrinsic structure of individual messages. Proc 12th USENIX Workshop on Offensive Technologies, Article 8.

Kleber S, Kargl F, State M, et al., 2022. Network message field type clustering for reverse engineering of unknown binary protocols. Proc 52nd Annual IEEE/IFIP Int Conf on Dependable Systems and Networks Workshops, p.80-87. https://doi.org/10.1109/DSN-W54100.2022.00023

Le SQ, Lai YX, Wang YP, et al., 2024. An adaptive classification and updating method for unknown network traffic in open environments. *Comput Netw*, 238:110114. https://doi.org/10.1016/j.comnet.2023.110114

Lee C, Bae J, Lee H, 2018. PRETT: protocol reverse engineering using binary tokens and network traces. Proc 33rd IFIP Int Conf on ICT Systems Security and Privacy Protection, p.141-155. https://doi.org/10.1007/978-3-319-99828-2_11

Leita C, Mermoud K, Dacier M, 2005. ScriptGen: an automated script generation tool for Honeyd. Proc 21st Annual Computer Security Applications Conf, p.203-214. https://doi.org/10.1109/CSAC.2005.49

Li JC, Cheng G, Yang GQ, 2023. Private protocol reverse engineering based on network traffic: a survey. *J Comput Res Dev*, 60(1):167-190 (in Chinese). https://doi.org/10.7544/issn1000-1239.202110722

Lin YD, Lai YK, Bui QT, et al., 2020. ReFSM: reverse engineering from protocol packet traces to test generation by extended finite state machines. *J Netw Comput Appl*, 171: 102819. https://doi.org/10.1016/j.jnca.2020.102819

Ma RK, Zheng H, Wang JY, et al., 2022. Automatic protocol reverse engineering for industrial control systems with dynamic taint analysis. *Front Inform Technol Electron Eng*, 23(3):351-360. https://doi.org/10.1631/FITEE.2000709

Muškardin E, Aichernig BK, Pill I, et al., 2022. AALpy: an active automata learning library. *Innov Syst Softw Eng*, 18(3):417-426. https://doi.org/10.1007/s11334-022-00449-3

Natella R, 2022. STATEAFL: greybox fuzzing for stateful network servers. *Empir Software Eng*, 27(7):191. https://doi.org/10.1007/s10664-022-10233-3

Pan Y, Lin W, Zhu YF, 2023. Progressive active inference method of protocol state machine. *Chin J Netw Inform Secur*, 9(2):81-93 (in Chinese). https://doi.org/10.11959/j.issn.2096-109x.2023023

Pham VT, Böhme M, Roychoudhury A, 2020. AFLNET: a greybox fuzzer for network protocols. Proc IEEE 13th Int Conf on Software Testing Validation and Verification, p.460-465. https://doi.org/10.1109/ICST46399.2020.00062

Saied M, Guirguis S, Madbouly M, 2024. Review of artificial intelligence for enhancing intrusion detection in the internet of things. *Eng Appl Artif Intell*, 127:107231. https://doi.org/10.1016/j.engappai.2023.107231

Shahbaz MM, 2008. Reverse Engineering Enhanced State Models of Black Box Software Components to Support Integration Testing. PhD Dissemination, Grenoble Universities, Auvergne-Rhône-Alpes, France.

Shevertalov M, Mancoridis S, 2007. A reverse engineering tool for extracting protocols of networked applications. Proc 14th Working Conf on Reverse Engineering, p.229-238. https://doi.org/10.1109/WCRE.2007.6

Sun FH, Wang S, Zhang CR, et al., 2019. Unsupervised field segmentation of unknown protocol messages. *Comput Commun*, 146:121-130. https://doi.org/10.1016/j.comcom.2019.06.013

Sun FH, Wang S, Zhang HL, 2022. A progressive learning method on unknown protocol behaviors. *J Netw Comput Appl*, 197:103249. https://doi.org/10.1016/j.jnca.2021.103249

Székely G, Ládi G, Holczer T, et al., 2021. Protocol state machine reverse engineering with a teaching-learning approach. *Acta Cybern*, 25(2):517-535. https://doi.org/10.14232/actacyb.288213

Tang T, Lai YX, Wang YP, 2023. Relational reasoning-based approach for network protocol reverse engineering. *Comput Netw*, 230:109797. https://doi.org/10.1016/j.comnet.2023.109797

Wang C, Wu LF, Hong Z, et al., 2015. Domain-specific algorithm of protocol state machine active inference. *Comput Sci*, 42(12):233-239 (in Chinese). https://www.jsjkx.com/CN/Y2015/V42/I12/233

Wang XW, Lv KZ, Li B, 2020. IPART: an automatic protocol reverse engineering tool based on global voting expert for industrial protocols. *Int J Parall Emerg Distrib Syst*, 35(3):376-395. https://doi.org/10.1080/17445760.2019.1655740

Wang YP, Zhang ZB, Yao DF, et al., 2011. Inferring protocol state machine from network traces: a probabilistic approach. Proc 9th Int Conf on Applied Cryptography and Network Security, p.1-18. https://doi.org/10.1007/978-3-642-21554-4_1

Wang YP, Yun XC, Shafiq MZ, et al., 2012. A semantics aware approach to automated reverse engineering unknown protocols. Proc 20th IEEE Int Conf on Network Protocols, p.1-10. https://doi.org/10.1109/ICNP.2012.6459963

Wang YP, Yun XC, Zhang YZ, et al., 2022. A multi-scale feature attention approach to network traffic classification and its model explanation. *IEEE Trans Netw Serv Manage*, 19(2):875-889. https://doi.org/10.1109/TNSM.2022.3149933

Whalen S, Bishop M, Crutchfield JP, 2010. Hidden Markov models for automated protocol learning. Proc 6th Int Conf on Security and Privacy in Communication Systems, p.415-428. https://doi.org/10.1007/978-3-642-16161-2_24

Ye YP, Zhang Z, Wang F, et al., 2021. NetPlier: probabilistic network protocol reverse engineering from message traces. Proc Network and Distributed System Security Symp. https://doi.org/10.14722/ndss.2021.24531

Yu ZH, Liu ZQ, Cong XY, et al., 2024. Fuzzing: progress, challenges, and perspectives. *Comput Mater Continua*, 78(1):1-29. https://doi.org/10.32604/cmc.2023.042361

List of supplementary materials

1 Method

Algorithm S1 Session response state label complementation algorithm (SRSLCA)

Algorithm S2 Traffic deduplication algorithm (TDA)

Algorithm S3 Extended prefix tree acceptor construction algorithm (EPTACA)

Algorithm S4 Response-based query optimization algorithm (RQOA)

Table S1 Acronym comparison