From Theory to Practice: Blackbox Testing in an Industrial Telecom Environment

Darshana Das K†, Krishnahari P†, Libna Kuriakose T†, Parvathy C M†, Ezudheen P
Government Engineering College Thrissur, India

Rahul Gopinath
University of Sydney, Australia

Abraham Jacob
Mettle Networks, India

***Abstract*—Coverage-guided fuzzing has proven highly effective in uncovering software vulnerabilities. However, many industrial systems cannot be instrumented or inspected due to security, legal, or operational constraints. In such settings, blackbox test generation remains the only viable testing strategy.**

This paper presents a case study in collaboration with our industrial partner, Mettle Networks, where we were tasked with testing the protocol-handling subsystem of a virtualized packet processing engine without access to the source code or internal documentation. To overcome this constraint, we applied blackbox grammar inference techniques to reverse-engineer the input protocol accepted by the system, achieving an F1 score of 0.94 for the inferred protocol grammar. Our blackbox test campaign resulted in 33.6% code coverage, as measured by Mettle Networks after the test campaign concluded.

We discuss our testing methodology and the practical challenges encountered. This case study highlights current limitations of automatic testing frameworks when deployed in blackbox industrial environments, and offers actionable insights for improving their effectiveness in such settings.

***Index Terms*—blackbox testing, grammar inference, protocol**

I. Introduction

Many industrial systems work in critical areas where the performance of the software system and its safety are both key requirements. However, these are often conflicting goals, and improvements in one can often be accomplished only by compromising the other. We were faced with one such challenge when working with our industrial partner.

In early 2022, we were approached by Mettle Networks, an IP networking product company, to test the protocol-handling subsystem of their virtualized packet processing engine (PPE). Traditional packet processing engines are implemented in ASIC hardware, and software based packet processing engines need to be competitive in terms of performance. Indeed, a key feature of Mettle Networks's packet processing engine was its ultra-fast performance compared to their competitors.

To improve performance, of their system, the Mettle Networks engineers extensively profiled their execution pathways and found that a significant chunk of the execution time was spent on parsing, validating, and re-validating input records from different interfaces. Mettle Networks engineers implemented performance optimizations in input processing, balancing system efficiency with security considerations. This led to a desire for external validation to ensure no unintended vulnerabilities were introduced. Hence, Mettle Networks sought an extensive evaluation of their protocol handling subsystem to ensure the product remained free of vulnerabilities and to obtain strong, measurable, and repeatable security assurances.

Due to intellectual property concerns and operational constraints typical of startups in competitive markets, Mettle Networks opted for blackbox testing without sharing proprietary source code or enabling deep instrumentation. As is common in early-stage virtualized network platforms, the protocol-handling subsystem of the packet processing engine did not include integrated coverage instrumentation at the time of testing. This made direct estimation of test completeness challenging. To address this, Mettle Networks assigned a senior engineer to provide expert insights into functionality coverage, validate observed behavior, and help interpret any unexpected system responses during the test campaign.

These are indeed constraints common in highly competitive startup industry environments. Our investigation of existing automated testing techniques revealed limited options, with most state-of-the-art approaches relying on coverage-guided or instrumentation-guided fuzzing, none of which were feasible in our constrained environment. We then investigated blackbox approaches conforming to the following requirements:

1) The technique should be blackbox.
2) It should allow us to leverage existing documentation.
3) It should provide statistical guarantees on test coverage.

Investigating grammar inference techniques, we found the L^* algorithm [1] with PAC oracle [2] to be best suited for our project. Hence, we adapted the L^* algorithm to construct and verify models of the protocol server we were testing through systematic exploration of input-output relationships.

We began with an initial protocol hypothesis derived from examples provided by Mettle Networks and the official documentation of the subsystem, expressed as an automata. We then systematically explored the acceptable transitions from each state using traditional blackbox fuzzing techniques. The transitions were specified by individual IP packets, specified in a binary format. These were built by specifying the packets using Kaitai Struct specifications [3]. Our inferred protocol provides a guarantee of correctness within 10% error with 90% confidence according to PAC framework. Additional tests reveal an F1 score of 0.94 for the inferred protocol grammar.

† These authors contributed equally to this work.

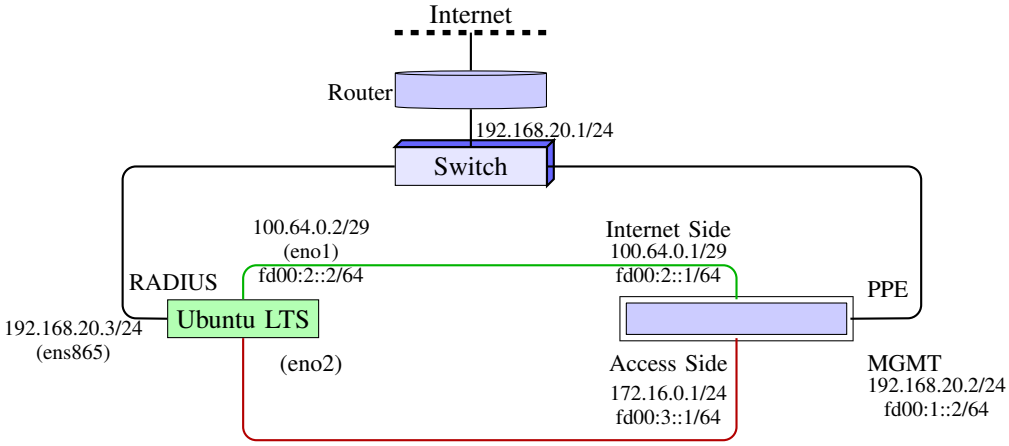

Fig. 1. Architectural overview of the protocol-handling subsystem

Our campaign identified several protocol implementation inconsistencies, which were validated by Mettle Networks. Subsequent evaluation of test coverage by Mettle Networks showed that our test cases achieved 33.6% coverage of the total code, which was judged as adequate by Mettle Networks.

This work demonstrates the effectiveness of combining automata learning with structured input generation for blackbox testing in industrial environments with access constraints.

Contributions:

- A practical methodology for applying blackbox grammar inference in industrial settings.
- Structured input generation for binary protocols using Kaitai Struct specifications.
- Identification of real-world constraints in applying grammar inference and fuzzing tools in industrial settings.
- Discovery of previously unknown protocol implementation inconsistencies in the protocol implementation.

II. Background

This section explains the background concepts.

Protocol Handling in Edge Packet Processing Systems. The protocol-handling subsystem under evaluation is part of a larger packet processing engine supporting edge networking use cases. In broadband environments, such systems often implement subscriber session control, authentication (e.g., via PPP over Ethernet as defined in RFC 2516 [4]), address management, policy enforcement, and session teardown.

This subsystem is organized into two logical planes: a control-plane, responsible for session signaling and communication with authentication servers (e.g., RADIUS), and a data-plane, responsible for forwarding encapsulated user traffic. Each plane adheres to specific message structures; for instance, control messages such as PADT and IPCP are handled by the control-plane, while encapsulated traffic flows through the data-plane. Message type determination is guided by fields such as the CODE field in PPPoE headers.

Automata Learning with L^* and PAC Guarantees. The L^* algorithm by Angluin [1], extended with the *Probably Approximately Correct* (PAC) framework [4], enables inferring a regular language from a blackbox system with statistical guarantees. The algorithm involves constructing a starting hypothesis for the regular language, and repeatedly refining it using membership queries to the blackbox. In the PAC setting, the language learned is within an ϵ approximation of the true language, with confidence $1 - \delta$.

Structure-Aware Input Generation for Binary Protocols. The PPPoE protocol is implemented over binary packet formats, which makes mutation-based fuzzing ineffective. To address this, we used Kaitai Struct [3] to specify binary packet structures declaratively. Kaitai Struct supports tag–length–value fields, unions, and context-sensitive specifications, enabling us to define accurate models of various message types (e.g., `PADT`, `PADO`, `PADS`).

III. Methodology

Our methodology consisted of three phases: (1) initial hypothesis, (2) protocol exploration and model inference, and (3) statistical estimation of input coverage and robustness.

A. Initial Hypothesis Construction

We based our initial hypothesis (G^i) on the activity diagram supplied by Mettle Networks (Figure 2). Mettle Networks provided us with a system running the packet processing engine configured with a RADIUS server, to which we could send packets using a client machine, also supplied by Mettle Networks. That is, the only interface we had access to was the DUT. We then refined it with available documentation including RFC2516 [5] detailing PPPoE, RFC1661 [6] detailing PPP. Our initial hypothesis had the following components:

- A set of *individual request and response messages* that constituted the alphabet of the system. Each message corresponded to a specific type of IP packet.
- Extracted from Figure 2, a preliminary list of high-level states in the PPPoE subsystem, and a short list of IP packet sequences that reaches each of these states, followed by a short list of IP packets that complete the system interaction ending in $\langle$*Termination*$\rangle$. This allowed us to construct the *observation table* required by L^*.

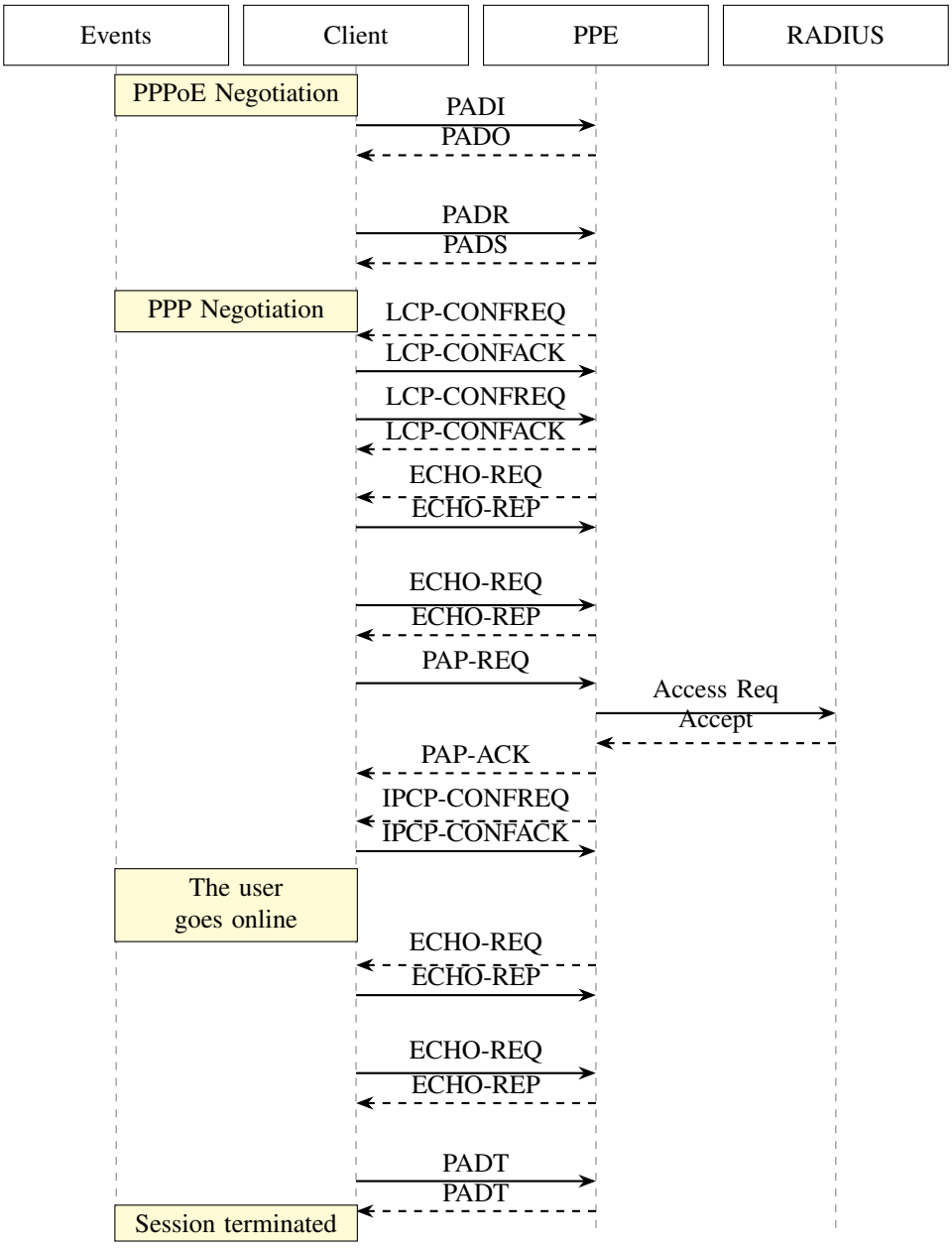

Fig. 2. PPPoE Authentication Sequence according to the available documentation.

Following the L^* algorithm, we constructed an **observation table** (called OT). As per L^*, the rows in the table represent protocol states (S), identified by the sequence of packets required to reach that state, and columns represent sequences of packets that can form packet sequence suffixes from these states, ending with termination of the interaction. Then we made them *closed* and *consistent* based on the L^* algorithm. For *closing* the table, we extended the OT with all transitions from each state as per L^*. Then we verified *consistency* by checking for duplicate rows, and verifying their transitions.

To generate IP packets, we manually constructed Kaitai Struct specifications for each packet type. These Kaitai Struct specifications were used to generate syntactically valid packets, and to parse and verify server response packets.

B. Protocol Exploration Phase

In this phase, we implemented a learning loop in which:

- The *learner* proposed a hypothesis grammar from the OT.
- The *system under test* was treated as an oracle, responding to membership queries (i.e., sequences of packets) with observable outputs such as response message types (However, in many cases, the system simply ignored unexpected packets) or connection resets. We could also periodically access the DUT logs, which indicated any system restarts (likely due to a abnormal program termination). We also noted that every client message had a server acknowledgment. For example, `C:PADI` & `S:PADO`, and `C:PAP-REQ` & `S:PAP-ACK`

⟨*PPPoE Protocol*⟩ → ⟨*Discovery*⟩ ⟨*Session*⟩ ⟨*Termination*⟩

⟨*Discovery*⟩ → `C:PADI` `S:PADO` `C:PADR` `S:PADS`

⟨*Termination*⟩ → `C:PADT` `S:PADT` | `S:PADT` `C:PADT`

⟨*Session*⟩ → ⟨*LCP State1*⟩ ⟨*Echo Seq*⟩ ⟨*PAP Auth*⟩ ⟨*IPCP State1*⟩ ⟨*Echo Seq*⟩ `DATA`
| ⟨*LCP State1*⟩ ⟨*Echo Seq*⟩ ⟨*Failed PAP Auth*⟩

⟨*LCP State1*⟩ → `S:LCP-CONFREQ` `C:LCP-CONFACK` ⟨*LCP State2*⟩
| `C:LCP-CONFREQ` `S:LCP-CONFACK` ⟨*LCP State1*⟩
| `C:LCP-CONFREQ` `S:LCP-CONFNACK` ⟨*LCP State1*⟩
| `S:LCP-CONFREQ` `C:LCP-CONFNACK` ⟨*LCP State1*⟩

⟨*LCP State2*⟩ → `C:LCP-CONFREQ` `S:LCP-CONFACK` ⟨*LCP State2*⟩
| `C:LCP-CONFREQ` `S:LCP-CONFNACK` ⟨*LCP State2*⟩
| `C:LCP-CONFREQ` `S:LCP-CONFACK`
| `C:LCP-CONFREQ` `S:LCP-CONFNACK`
| ϵ

⟨*Echo Seq*⟩ → `C:ECHO-REQ` `S:ECHO-REP` ⟨*Echo Seq*⟩
| `S:ECHO-REQ` `C:ECHO-REP` ⟨*Echo Seq*⟩
| ϵ

⟨*PAP Auth*⟩ → `C:PAP-REQ` `S:PAP-ACK`

⟨*Failed PAP Auth*⟩ → `C:PAP-REQ` `S:PAP-NACK`

⟨*IPCP State1*⟩ → `C:IPCP-CONFREQ` `S:IPCP-CONFACK` ⟨*IPCP State1*⟩
| `C:IPCP-CONFREQ` `S:IPCP-CONFNACK` ⟨*IPCP State1*⟩
| `S:IPCP-CONFREQ` `C:IPCP-CONFACK` ⟨*IPCP State2*⟩
| `S:IPCP-CONFREQ` `C:IPCP-CONFNACK` ⟨*IPCP State1*⟩

⟨*IPCP State2*⟩ → `C:IPCP-CONFREQ` `S:IPCP-CONFACK` ⟨*IPCP State2*⟩
| `C:IPCP-CONFREQ` `S:IPCP-CONFNACK` ⟨*IPCP State2*⟩
| `C:IPCP-CONFREQ` `S:IPCP-CONFACK`
| `C:IPCP-CONFREQ` `S:IPCP-CONFNACK`
| ϵ

Fig. 3. The PPPoE protocol, as discovered by our testing. The client requests are prefixed with `C:` and the server responses are prefixed with `S:`.

or `S:PAP-NACK`. Our input distribution was based on fuzzing and mutation. Essentially, we generated a packet sequence using the hypothesis grammar, and mutated this sequence by adding deleting or substituting a random element in the sequence, following traditional fuzzing techniques, producing an incorrect packet sequence, and observing the result. If the system accepted the packet sequence (i.e., the user was logged in, and we get a `S:IPCP-CONFACK` response), this was then a counter example. If the system did not respond as expected, we logged this as a potential protocol implementation error. The queries required for PAC guarantee is computed as: $N \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$ We fixed the PAC parameters to $\epsilon = 0.1$ and $\delta = 0.1$ resulting in $N \geq 93$. This allowed us to confirm the correctness of the inferred grammar with $(1-\delta) = 90\%$ confidence and $\epsilon = 10\%$ error. When this resulted in a surprising output, this was taken as a new trace, resulting in identification of a new state.

- We updated the OT based on the L^* algorithm, following *closedness* and *consistency* requirements.

The loop continued until the hypothesis stabilized and no new distinguishable states were discovered. This iterative refinement process confirmed the absence of undocumented transitions or protocol extensions within the scope of observed interactions. The final grammar (G^f) obtained at the end of exploration is given in Figure 3. Note that the grammar does not capture the complete behavior observed (see Section V).

C. Obtaining Reliability Guarantees

In this final phase, we sought to provide formal and statistical guarantees regarding the completeness of our testing campaign and the robustness of the PPPoE implementation. While δ and ϵ provides theoretical accuracy and confidence for learned grammar, we wanted to verify the practical accuracy and precision achieved. To assess the syntactic correctness of the inferred grammar, we evaluated the $F1$ score, as follows: We used the inferred protocol grammar to compute a sequence of packets, and checked if the system accepted this packet sequence. Any that were not accepted by the system based on the next packet from the system either not sent, or unexpected was marked as *False Positive* (FP), and others as *True Positive* (TP). We next picked a random client request in the generated input sequence, and replaced it with another random client request that is not in the grammar. Then, checked if the blackbox accepted this new sequence. If the new sequence was accepted, then it is *False Negative* (FN). If not, it is *True Negative* (TN). *Precision* (P) is computed as $\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$ and *recall* (R) is computed as $\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$. The F1 score is computed as: $\mathrm{F1} = 2 \times \frac{\mathrm{P \times R}}{\mathrm{P+R}}$.

To assess the structural completeness of the inferred grammar, we generated test cases from the inferred protocol grammar using *k-path coverage* [7], ensuring that all distinct execution paths up to length $k = 3$ were exercised at least once. We then requested Mettle Networks to provide us with the path coverage obtained for the test cases thus generated.

This methodology enabled us to construct a robust, input coverage-driven test suite despite the blackbox nature and binary complexity of the system under test. The final inferred grammar for PPPoE protocol is in Figure 3.

IV. Evaluation and Results

For evaluation, we used a system with the packet processing engine installed, with a client that connected to the engine.

Grammar Accuracy (δ) and Confidence (ϵ). Due to the testing procedure adopted, we provide a statistical guarantee that our inferred protocol in Figure 3 is within 10% approximation of the true protocol implemented with a confidence of 90%. During testing, we obtained a precision of 0.93, recall of 0.94, resulting in f1 = 0.94.

Protocol coverage. Due to the testing procedure adopted, we have covered the protocol grammar to a depth of $k-\text{path} = 3$.

Code coverage. To check the effectiveness of our test campaign, Mettle Networks instrumented their system, and extracted the code coverage achieved. We provided them with the test cases resulting from $k-\text{path} = 3$, which was run manually by Mettle Networks. Mettle Networks determined that the code coverage achieved was 33.6% of the total codebase, which was judged as *adequate* given the blackbox constraints and asynchronous nature of the protocol.

V. Discussion

We next discuss the challenges involved in this campaign.

Asynchronous behavior. One of the main challenges that we faced was the asynchronous nature of the PPPoE protocol. For example, We found that during ⟨*Session*⟩, `ECHO-REQ` and `ECHO-REP` packets may be sent periodically to detect link failure. Either the server or the client may send these requests in between processing. The server or client may skip a response if another request or response is sent instead. This cannot be captured in a formal grammar. Next, the server or the client may send a `PADT` request at any point after the ⟨*Discovery*⟩ phase to terminate the connection. This is very difficult to capture cleanly in the formal grammar. The server has a timeout mechanism. However, it always sends a `PADT` packet before it times out.

Furthermore, during ⟨*Session*⟩ phase, when the server receives an acknowledgment (`LCP-CONFACK`) for its configuration request (`LCP-CONFREQ`), it allows the session to proceed to authentication (⟨*Auth X*⟩), even if the configuration request from client does not receive an acknowledgment. The server also can sends as many `LCP-CONFACK` packets to the number of `LCP-CONFREQ` packets it receives from the client. That is, only the count may be equal. The client need not send any `LCP-CONFREQ` packet, but the server always sends a `LCP-CONFREQ`, and if not acknowledged, will send a `PADT` at timeout. A similar pattern is observed during the IPCP configuration phase. During ⟨*Echo Seq*⟩, a `ECHO-REP` is implied if the next packet (the authentication) is sent.

Inability to identify system state. In several points in the interaction, one may send a malformed packet, and the system can choose to ignore the packet it does not validate. However, in other cases, it may not validate the packet, which may put it in an unexpected state. It was not easy to distinguish between these two different states.

Server initiated requests. The protocol also allows the server to initiate requests, and these requests can be interleaved with the requests being sent from the client. This could not be captured in the grammar.

Single client. As this was the initial investigation for feasibility of blackbox testing, we were allocated only a single client for exploration, which made it difficult to check for other system states like non responsive system—it was difficult to check if the requests from a single client was being held up, or if the entire system was being non responsive.

Code coverage. Onsite evaluation determined that code coverage from $k-$path $=3$, was 33.6%, demonstrating that a significant chunk of the functionality was based on deeper logic. While we probably reached larger coverage during fuzzing (as demonstrated by implementation errors detected), the coverage achieved by fuzzer could not be extracted.

We note that code coverage was evaluated only from a selected set of static test cases that we supplied. This in no way reflects the actual testedness of the product as Mettle Networks has its own internal testing infrastructure that tests its product to a deeper level.

Mettle Networks also asked us to focus on IPv4, even though the server supported IPv6, which required a different packet structure and was therefore not exercised.

Protocol Handling Observations and Improvements. We discovered several inconsistencies in protocol implementation, as confirmed by Mettle Networks.

1) Certain non-standard packet sequences led to unexpected behavior, which Mettle Networks recognized and is addressing through ongoing improvements.
2) Testing uncovered an optimization opportunity in handling certain protocol messages under abnormal conditions, which Mettle Networks is actively resolving to enhance system resilience.
3) Testing revealed opportunities to strengthen protocol validation, particularly in handling certain authentication message variants, which Mettle Networks is incorporating into its development road-map.
4) Similarly, the server responded with an acknowledgment for the authentication requests when the PAP Code was set to other reserved values (e.g., the code corresponding to Authenticate-Nak), provided the rest of the packet resembled a valid request.
5) Certain edge cases in protocol handling were identified where stricter address validation can improve robustness. Mettle Networks is implementing updates to enforce tighter protocol compliance.

These findings highlight gaps in protocol validation, which could lead to security vulnerabilities and system instability.

VI. Related Works

Several grammar inference algorithms exist that extract context-free grammars from inputs. These include: GLADE [8], ARVADA [9], and TreeVada [10]. Kedavra [11]. The main reason we were unable to use these is due to our requirement for (1) ability to start with an initial hypothesis and (2) statistical guarantees in the resulting grammar.

Several works target protocols under blackbox constraints: 5GC-Fuzz [12] 5G core network implementations, Exploiting Dissent [13], and Protocol State Fuzzing [14] that focuses on TLS implementations.

Two that comes closest to our work are: PULSAR [15] combines fuzzing with automatic protocol reverse engineering to infer message formats and protocol states for uncovering deep vulnerabilities in closed-source network protocol implementations. SBEPFuzz [16] uses Monte Carlo tree search algorithm for stateful black-box fuzzing of protocol implementations.

VII. Actionable Insights

Our industrial case study revealed several practical challenges and lessons for both practitioners developing network protocols and researchers working on blackbox testing.

A. Limitations in Functionality Coverage

Despite systematic exploration using L^* learning, we achieved only a limited coverage of the total codebase, leaving much of deeper functionality unexplored. This limitation stemmed from several factors:

Note from Mettle Networks: At Mettle Networks, we view collaborations with academic researchers as an essential part of our commitment to engineering excellence. This engagement exemplifies how cutting-edge formal methods can be integrated into industrial workflows to enhance protocol robustness and validation strategies. The insights and techniques demonstrated through this collaboration are helping us evolve our development and testing pipelines to meet the demands of increasingly complex networking systems. We see such partnerships as a vital mechanism to go from good to great—not only validating correctness but also shaping the future of verifiable network systems. This work stands as a shared success story, showcasing how research and industry can co-create tools and techniques that raise the bar for infrastructure reliability.

Alphabet Dependency: Grammar inference is fundamentally constrained by alphabet diversity. Our IPv4-focused packets prevented IPv6 exploration, demonstrating that alphabet breadth directly determines discoverable behavior coverage.
Surface-Level Protocol Coverage: Despite complete protocol coverage, deeper functionalities (QoS policies, routing decisions, edge-cases) remained unexercised, likely requiring specific, difficult-to-trigger network conditions.

B. Recommendations for Protocol Testing

Comprehensive Seed Corpus: Capture diverse real-world packet traces spanning the system's operational envelope, including edge cases for complete alphabet construction.
Multi-Protocol Initialization: Include representatives from all supported protocol versions (IPv4/IPv6, authentication methods) in initial hypothesis for comprehensive exploration.

C. Design Recommendations for Testable Systems

Minimize Asynchronous Behavior: Asynchronous interactions complicated testing and caused protocol implementation inconsistencies. Avoid asynchronous behavior unless performance-critical, or implement clear timeout/synchronization mechanisms.
Explicit Error Responses: Ignoring ill-formed packets hampered testing. Design systems to respond to ill-formed packets with explicit errors during development.

D. Implications for Blackbox Testing Research

Our experience highlights several areas where current blackbox testing techniques could be enhanced:
Alphabet Discovery: Current grammar inference approaches assume a known alphabet. Developing techniques for automatically discovering or expanding the input alphabet during testing would significantly improve the practical applicability.
Asynchronous Protocol Handling: Grammar inference methods struggle with asynchronous protocols. Research into learning algorithms that can handle non-deterministic timing and out-of-order responses would broaden their applicability.

VIII. Conclusion

We present a case study in applying blackbox testing techniques to a performance-critical, proprietary telecom system with significant access constraints. We were tasked with evaluating a packet processing system without access to source code, instrumentation, or detailed protocol documentation.

To address this challenge, we combined automata learning using L^* algorithm and request and response formats described by Kaitai Struct to infer a model of the protocol's state machine and systematically explore its behavior. Our inferred protocol provides a guarantee of correctness of 10% with 90% confidence according to the PAC framework, and an F1 score of 0.94 for the inferred protocol grammar.

Our methodology uncovered previously unknown vulnerabilities and correctness issues, which were confirmed and addressed by the industrial partner Mettle Networks. More broadly, our experience highlights several limitations of current research tools when applied in real-world, uninstrumented environments, particularly those using binary protocols.

This work demonstrates the practicality of combining formal learning with binary-aware test generation for blackbox testing in industrial contexts. As more companies prioritize performance and IP protection, such techniques are likely to become increasingly valuable.

References

[1] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.
[2] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
[3] Kaitai Project, "Kaitai struct." https://kaitai.io/, 2025.
[4] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[5] L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, and R. Wheeler, "A method for transmitting ppp over ethernet (pppoe)." https://www.rfc-editor.org/info/rfc2516, Feb. 1999. RFC 2516, Informational.
[6] W. Simpson, "The point-to-point protocol (ppp)." https://www.rfc-editor.org/info/rfc1661, July 1994. RFC 1661, Internet Standard (STD 51).
[7] N. Havrikov, A. Kampmann, and A. Zeller, "From input coverage to code coverage: Systematically covering input structure with k-paths," *TOSEM*, 2022.
[8] O. Bastani, R. Sharma, A. Aiken, and P. Liang, "Synthesizing program input grammars," *ACM SIGPLAN Notices*, 2017.
[9] N. Kulkarni, C. Lemieux, and K. Sen, "Learning highly recursive input grammars," in *ASE*, IEEE, 2021.
[10] M. R. Arefin, S. Shetiya, Z. Wang, and C. Csallner, "Fast deterministic black-box context-free grammar inference," in *ICSE*, 2024.
[11] F. Li, X. Chen, X. Xiao, X. Sun, C. Chen, S. Wang, and J. Han, "Incremental context-free grammar inference in black box settings," in *ASE*, 2024.
[12] G. Nakas, P. Radoglou-Grammatikis, G. Amponis, T. Lagkas, V. Argyriou, S. Goudos, and P. Sarigiannidis, "5g-fuzz: An attack generator for fuzzing 5gc, using generative adversarial networks," in *IEEE Globecom Workshops*, IEEE, 2023.
[13] A. Walz and A. Sikora, "Exploiting dissent: towards fuzzing-based differential black-box testing of tls implementations," *IEEE TDSC*, 2017.
[14] J. De Ruiter and E. Poll, "Protocol state fuzzing of {TLS} implementations," in *USENIX Security*, pp. 193–206, 2015.
[15] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Pulsar: Stateful black-box fuzzing of proprietary network protocols," in *EAI International Conference, SecureComm*, Springer, 2015.
[16] J. Guo, C. Gu, X. Chen, X. Zhang, K. Tian, and J. Li, "Stateful black-box fuzzing for encryption protocols and its application in ipsec," *Computer Networks*, vol. 251, p. 110605, 2024.