Active vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols

Bernhard K. Aichernig[1]
[1]Institute of Software Technology
Graz University of Technology
Graz, Austria
aichernig@ist.tugraz.at

Edi Muškardin[1,2]
[2]Silicon Austria Labs
TU Graz - SAL DES Lab
Graz, Austria
edi.muskardin@silicon-austria.com

Andrea Pferscher[1]
[1]Institute of Software Technology
Graz University of Technology
Graz, Austria
andrea.pferscher@ist.tugraz.at

Active automata learning became a popular tool for the behavioral analysis of communication protocols. The main advantage is that no manual modeling effort is required since a behavioral model is automatically inferred from a black-box system. However, several real-world applications of this technique show that the overhead for the establishment of an active interface might hamper the practical applicability. Our recent work on the active learning of Bluetooth Low Energy (BLE) protocol found that the active interaction creates a bottleneck during learning. Considering the automata learning toolset, passive learning techniques appear as a promising solution since they do not require an active interface to the system under learning. Instead, models are learned based on a given data set. In this paper, we evaluate passive learning for two network protocols: BLE and Message Queuing Telemetry Transport (MQTT). Our results show that passive techniques can correctly learn with less data than required by active learning. However, a general random data generation for passive learning is more expensive compared to the costs of active learning.

1 Introduction

Behavioral models provide a useful tool for the analysis and verification of autonomous systems. However, the availability of such a model might be limited. Manually creating a model is tedious and even if a model exists, keeping it up-to-date presents an ongoing challenge. Cyber-physical systems might consist of many heterogeneous components, e.g. sensors, which are accessible communication protocols. However, access to these components or to the protocol implementations might be restricted, especially in third-party components. Such an environment motivates the automatic generation of a behavioral model from components of an autonomous system.

Automata learning has successfully been applied to extract behavioral models out of black-box systems. Besides theoretical research and learning competitions [20, 9, 19], nowadays automata learning is successfully applied in practice. In recent years, research has focused on learning behavioral models of network protocols like BLE [28], (D)TLS [32, 13], MQTT [36], or TCP [12]. All of them applied active automata learning to generate a behavioral model of an implementation of the tested network protocol.

Active learning characterizes that the behavior is explored by active interaction with the System Under Learning (SUL). Depending on the application area, the development of a learning setup that enables active querying might be a tedious process. For example, consider the learning of wireless network protocols: sent packets might get lost or arrive delayed. Such behavior could introduce non-determinism.

M. Luckcuck and M. Farrell; M. Autili, L. Berardinelli,
A. Bucaioni and C. Pompilio (Eds.):
Formal Methods for Autonomous Systems and
Automated and verifiable Software sYstem DEvelopment
EPTCS 371, 2022, pp. 1–19, doi:10.4204/EPTCS.371.1

Non-deterministic behavior might interfere with the desired learning algorithm, which usually requires the SUL to behave deterministically. To treat non-deterministic behavior, the implementation of fault-tolerant mechanisms is necessary. Such mechanisms [28, 13] include, e.g., the repetition of queries or the introduction of a cache to determine the correct output for a repeated input.

In automata learning, we distinguish between two learning paradigms: active and passive learning. These two paradigms are different in the generation of the required data set from which the behavioral model is inferred. As outlined before, active algorithms interact with the SUL to generate the required learning data. In contrast, passive techniques use a given data set, e.g. log files, to generate a behavioral model. Consequently, passive learning algorithms can only cover the behavior that the given data includes. However, passive learning techniques might be better suited if the development of a sufficient active learning setup is costly in terms of the effort required to enable fault tolerance.

The previous work [32, 36, 12, 28] argues that active learning provides in-depth behavioral aspects of the SUL. However, none of the techniques have been compared with passively generated data. We raise the question if randomly generated data is good enough to cover important behavioral aspects of the SUL. Furthermore, we approach the reduction of the data set size required for correct passive learning.

In this paper, we will compare passive and active techniques for the learning of Bluetooth Low Energy (BLE) devices and Message Queuing Telemetry Transport (MQTT) protocol implementations. In previous work [28], we discussed the difficulties in the wireless learning of BLE protocol implementations on different devices via a handcrafted interface. One of the main challenges was to guarantee that the device was sufficiently reset before performing the next query. Furthermore, the results show that timing constraints might differ between the BLE devices. Tappler et al. [36] faced similar challenges in learning the MQTT protocol. They also required individual learning setups for different MQTT server implementations. In passive learning, these challenges do not need to be handled during learning. Hence, we are interested to see if passive learning can achieve similar results to active learning.

The aim of this paper is not only to discuss whether an active or a passive learning technique is preferable. In addition, we want to approach the challenge of generating a minimal adequate sample for both learning paradigms. We denote the optimization of data by minimizing the sample. Based on these problems, we discuss the following three research questions:

(RQ 1) Can passive learning based on a random sample outperform active learning?

(RQ 2) Does the considered active automata learning algorithm generate an optimal sample?

(RQ 3) Can random sampling support active automata learning?

Structure. The paper is structured as follows. In Sect. 2, we introduce background concepts of our performed evaluation. Section 3 introduces the methodology and presents the result of the performed case study. We show related work in Sect. 4. Finally, Sect. 5 concludes the paper with a summary, a discussion of the found results, and an outlook on future work.

2 Background

2.1 Mealy machine

Mealy machines represent a modeling formalism for the behavioral description of reactive systems. A Mealy machine is a finite state machine with transitions labeled with an input action and the corresponding observable output. We define a Mealy machine $\mathscr{M}$ as a 6-tuple $\langle Q, q_0, I, O, \delta, \lambda \rangle$ where Q is the finite state set, q_0 is the initial state, I is the finite set of inputs, O is the finite set of outputs, $\delta : Q \times I \to Q$ is the state transition function, and $\lambda : Q \times I \to O$ is the output function.

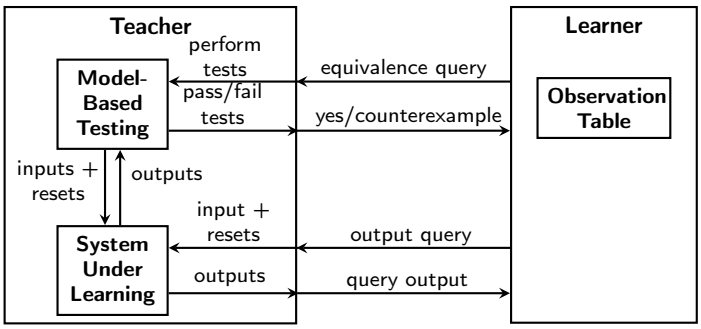


Figure 1: Angluin's [5] adapted Minimally Adequate Teacher (MAT) framework. The adaptions relate to learning of reactive systems based on Smeenk et al. [34] and Tappler et al. [1].

We assume that $\mathscr{M}$ is input-enabled. Hence, for every $q \in Q$ and $i \in I$, $\delta(q,i)$ and $\lambda(q,i)$ are defined. Furthermore, the definition of δ as a non-set-function implies that $\mathscr{M}$ is deterministic. Let $|S|$ denote the number of elements in a set S. We denote the size of $\mathscr{M}$ by the number of states, i.e. $|Q|$.

A sequence of input/output pairs is denoted as $s \in (I \times O)^*$, where $s^I \in I^*$ and $s^O \in O^*$ are the corresponding input and output sequences. We write $s \cdot s'$ for the concatenation of two sequences. Additionally, we write ε for the empty sequence and also lift a single element $e \in (I \times O)$ to sequences. Let $|s|$ express the number of input/output pairs in a sequence. We extend δ and λ for sequences. The function $\delta^* : Q \times I^* \to Q$ returns the state reached after executing the given input sequence and $\lambda^* : Q \times I^* \to O^*$ returns the observed output sequence. An execution on $\mathscr{M}$ returns the output sequence of an input sequence starting from the initial state, i.e. $s^O = \lambda^*(q_0, s^I)$. A *trace* is an input/output sequence of an execution on $\mathscr{M}$. Let T be the set of traces that can be generated from executions on a Mealy machine $\mathscr{M}$. We denote two Mealy machines $\mathscr{M}$, $\mathscr{M}'$ as equivalent if they generate the same set of traces T.

2.2 Automata Learning

In automata learning, we infer a behavioral model from the system's data where this data could be given log files or actively queried. Generally, we assume that the SUL is a black box, where no insight into internal behavior is provided. Depending on the generation of data for learning, we distinguish between two paradigms: active and passive learning.

2.2.1 Active Automata Learning

Active automata learning actively queries the SUL to infer a behavioral model. For this, the algorithms require an interface that enables the querying of input sequences.

Many active learning algorithms build on Angluin's L^* algorithm [5]. In her seminal work, she introduced an algorithm that learns a Deterministic Finite Automaton (DFA) representing an unknown regular language. For this, she introduced the Minimally Adequate Teacher (MAT) framework. Figure 1 depicts a modified version of Angluin's MAT framework. The MAT framework distinguishes two members: the learner and the teacher. The teacher knows the regular language that is depicted as SUL and the learner wants to learn a behavioral model of the SUL.

The L^* algorithm is an iterative procedure where each iteration includes two stages. According to the current stage, the learner asks two different kinds of questions: membership queries and equivalence queries. In the first stage, the learner aims to generate a hypothesis of the SUL. The hypothesis is

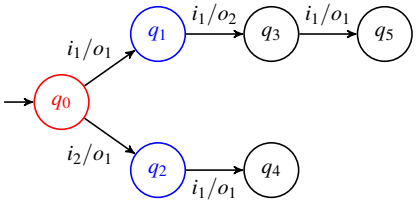
Table 1: Observation table for Mealy machines from Shahbaz and Groz [33] with a modified E set.

		E		
		i_1	i_2	$i_1 \cdot i_1$
S	ε	o_1	o_1	$o_1 \cdot o_2$
$S \cdot I$	i_1	o_2	o_1	$o_2 \cdot o_1$
	i_2	o_1	o_1	$o_1 \cdot o_2$

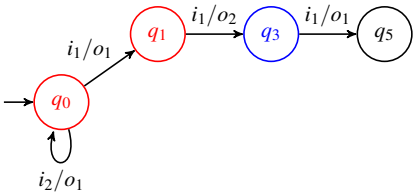
a conjecture about the behavioral model of the System Under Learning (SUL). For the creation of the hypothesis, the learner asks if a word is included in the regular language. The teacher answers these membership queries either by *yes* or *no*. The learner collects the answers of the teacher in an observation table. Based on the adequately filled observation table, the learner constructs a hypothesis. This hypothesis is then proposed to the teacher, who checks the equivalence to the SUL. The equivalence check between the provided hypothesis and SUL is considered the second stage of the learning algorithm. If they are equivalent, the teacher answers with *yes* and the learning algorithm terminates by returning the proposed hypothesis. Otherwise, the teacher provides a counterexample to the conformance between the hypothesis and SUL, and the algorithm returns to the first stage. In the next iteration, the provided counterexample is then taken by the learner to retrieve new membership queries and to construct a new hypothesis. Hence, one iteration in active automata learning includes in the first stage the posing of several membership queries and in the second stage the equivalence check. We call one iteration *learning round*. This iterative procedure is repeated until a conforming model is found.

The L^* algorithm and the MAT framework have been extended for other modeling formalism. Shahbaz and Groz [33] inferred Mealy machines from reactive systems. For this, they modified the original MAT framework as follows: Instead of asking membership queries, output queries are posed. An output query is an input sequence that the teacher executes on the SUL and then returns the corresponding output sequence. Again, the learner stores the received output sequences in the observation table. For learning Mealy machines, the observation table can be described with a triplet $\langle S, E, T \rangle$, where S and E are non-empty sets of input sequences and $T : (S \cup S \cdot I) \times E \to O^*$ is a function with I and O as input and output set of the Mealy machine. S is prefix-closed and E is suffix-closed. According to Shahbaz and Groz [33], S is initialized with the empty sequence ε and E with the inputs I. Table 1 shows an extended example of an observation table taken from Shahbaz and Groz [33]. The columns of an observation table are defined by E, and the rows by S and $S \cup I$. The function T accesses an output sequence stored in a table cell. A table row can be accessed via entries of $S \cup S \cdot I$ and is defined by the outputs of the corresponding values of E. The outputs in the cell $T(s,e)$ correspond to the outputs observed on the execution of $\lambda^*(\delta^*(q_0,s),e)$ with $s \in S \cup S \cdot I$ and $e \in E$. For example, the learner wants to query the output for a cell accessed by $T(i_2, i_1 \cdot i_1)$. For this, the learner asks the teacher the following output query: $i_2 \cdot i_1 \cdot i_1$, and the teacher responds with the query output $o_1 \cdot o_1 \cdot o_2$. The output sequence $o_1 \cdot o_2$ of the E set entry $i_1 \cdot i_1$ is then stored in the row of the observation table indexed by i_2. We refer to Shahbaz and Groz [33] for a detailed description of creating a Mealy machine from the observation table.

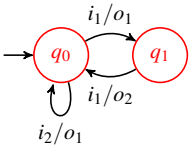
The application of automata learning is limited by the assumption of a perfect teacher that can determine the equivalence between a hypothesis and the SUL. In practice, we substitute the equivalence oracle with model-based testing techniques as shown in Fig. 1. Tretmans [37] introduces an implementation relation $\mathfrak{I}$ **imp** $\mathfrak{S}$ which defines that an implementation $\mathfrak{I}$ conforms to a specification $\mathfrak{S}$. In conformance testing, we want to find a test suite $T_{\mathfrak{S}}$ that can assess for every implementation $\mathfrak{I}$ if it implements a specification $\mathfrak{S}$. The aim of conformance testing in learning is to create a test suite that enables the assessment if a provided hypothesis $\mathfrak{H}$ conforms to implementation $\mathfrak{I}$ which represents our SUL. We say that the implementation $\mathfrak{I}$ passes a test sequence $t \in T_{\mathfrak{H}}$ $\mathfrak{I}$ **passes** t, if the output sequence generated from



(a) Initial PTA generated from the two given traces in Sect. 2.2.2. Blue nodes indicate candidates for merging with red nodes.



(b) Automaton after merging node q_2 into q_0 of the PTA.



(c) Final merged automaton for the given traces.

Figure 2: Steps of the RPNI algorithm starting from the initial PTA to the final merged automaton.

the execution on the learned model $\mathfrak{H}$ is equal to the obtained outputs sequence on $\mathfrak{I}$. Based on Tretmans implementation relation, we define conformance by the following equation.

$$\mathfrak{I} \textbf{ imp } \mathfrak{H} \iff \forall t \in T_{\mathfrak{H}} : \mathfrak{I} \textbf{ passes } t. \tag{1}$$

The objective of conformance testing during learning is to find a test sequence that violates Eq. 1. A found counterexample is then used to refine the hypothesis. In practice, the challenge of active automata learning is the selection of a conformance testing technique that is efficient in the number of required interactions with SUL, but also effective in the sense of finding counterexamples to the conformance between the hypothesis and the SUL.

2.2.2 Passive Automata Learning

Passive automata learning infers an automaton from a given data set. Gold showed that the problem of inferring a DFA with k states from a given data set is *NP-complete* [16]. State merging is one of the key technologies that is used in passive learning algorithms like in the Regular Positive Negative Inference (RPNI) [26, 17] algorithm. RPNI takes a set of positive and negative traces. Positive traces include behavior that should be described by the learned automaton, whereas the behavior shown in negative traces must not be included. Based on the positive traces, RPNI builds a Prefix Tree Acceptor (PTA). States of the PTA are then merged to create a finite automaton. A merge is valid if no negative trace can be generated by the merged automaton. Otherwise, if the merged automaton now includes negative examples, the merge is dismissed and a different merge is attempted.

Variants of the RPNI algorithm can learn Mealy machines from input/output traces. For Mealy machines, states are merged iff outputs for a corresponding input are similar. For example, consider the following two traces:

$$\begin{array}{lllllll} (1) & (i_1, & o_1) & \cdot & (i_1, & o_2) & \cdot \quad (i_1, \quad o_1) \\ (2) & (i_2, & o_1) & \cdot & (i_1, & o_1) & \end{array}$$

The PTA for these two traces is shown in Fig. 2a. To generate a more general behavioral model, RPNI merges states. The red states indicate the states that will be included in the final automaton and the blue states are currently candidates for merging with red states. For example, in the PTA presented in Fig. 2a, we first check if we can merge the blue state q_1 into the red state q_0. This merge is not possible since the input i_1 generates different outputs (o_1 vs. o_2). However, q_2 can be merged into q_0. Figure 2b shows the automaton after merging. In the next step, q_3 can be merged into q_0, which leads to the final automaton presented in Fig. 2c. Note that passive learning algorithms can only model behavior that is included in

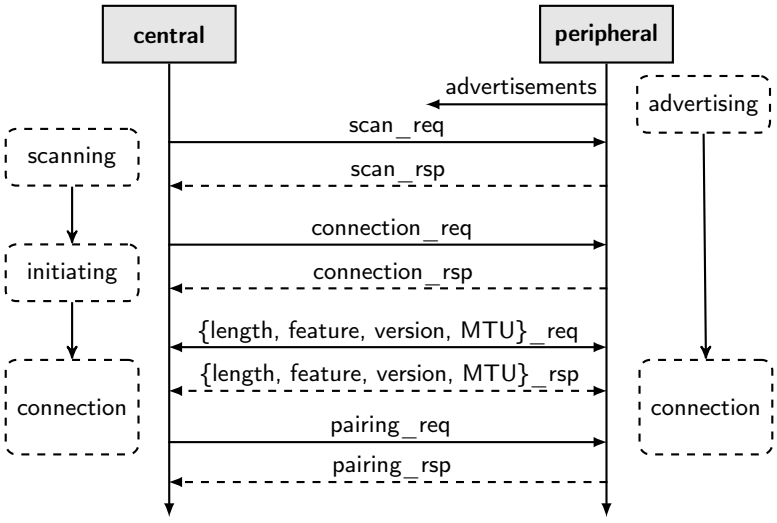


Figure 3: BLE sequence diagram for the establishment of a connection between two devices. The central device initiates the connection to the peripheral device. Note that after the initiation of the connection both devices could send requests that must be replied to by the other party. This figure is taken from [28].

the given data set. For example, the learned automaton presented in Fig. 2c is not input enabled, since no trace was given that described the behavior of input i_2 in state q_1.

2.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless communication protocol that has been introduced by Bluetooth Standard 4.0. Similar to the Bluetooth classic protocol, also known as Basic Rate, BLE is used for short-range peer-to-peer communication. However, BLE and Bluetooth classic are not compatible and depend on different implementations. The design of the BLE protocol also enables communication via Bluetooth for low-energy devices as they occur in the Internet of Things (IoT).

The Bluetooth standard [7] distinguishes between two communicating parties: the central and the peripheral device. For simplicity, we refer to the central device as *central* and to the peripheral device as *peripheral*. Figure 3 shows the sequence diagram of the connection procedure. The peripheral starts in the advertising state by sending advertisements to publicly announce that it is ready for a connection. The central device searches for advertisements by sending a scan request (scan_req), to which the peripheral responds either by an advertisement or a dedicated scan response. Both answers are abstracted as scan_rsp. Afterward, the central initiates the connection by sending a connection request (connection_req), which is responded by a connection response (connection_rsp). After the initiation of the connection, the central and the peripheral negotiate connection parameters. For example, the central can indicate the used Bluetooth version or request the maximum transmission unit (MTU). Both, the central and the peripheral, can send requests for specific parameters that should be valid for the current session. To further proceed to the pairing procedure these posed requests must be answered accordingly by the receiving party. After the negotiation phase, the pairing procedure can start. In the pairing procedure, keys for an encrypted communication are exchanged.

2.4 Message Queuing Telemetry Transport

The MQTT protocol [25] is a publish/subscribe protocol. Due to its lightweight design, it is also a popular protocol for applications in the IoT. The MQTT protocol distinguishes two members: the client and the broker. The broker is a central server unit that distributes messages from the clients. Clients connect to the broker and publish messages on a topic and the broker forwards then these published messages to clients who subscribed to the topic. In addition, some brokers offer advanced features like the definition of the client's will. The will of a client is a message assigned to a specific topic. The broker publishes the will of a client when the client disconnects. In the literature, we find several case studies [36, 1, 27] which applied active automata learning to learn behavioral models of different MQTT-broker implementations. The learned behavioral models revealed inconsistencies in the MQTT specification. Note that the used modeling formalism requires the SUL to be deterministic. To simulate a deterministic behavior, the used learning setups required individual configurations since timed behavior led to non-deterministic observations. The observed non-determinism motivates the MQTT case study for comparison to passive learning where non-determinism could be handled offline.

3 Evaluation

In the following, we present our conducted case study and investigate the following research questions:

(RQ 1) Can passive learning based on a random sample outperform active learning?

(RQ 2) Does the considered active automata learning algorithm generate an optimal sample?

(RQ 3) Can random sampling support active automata learning?

Firstly, we introduce our applied methodology. For this, we present the used learning setup and the case study subjects. Furthermore, we explain the applied assessment technique of the gained learning results. Secondly, we discuss different sampling techniques that are used to generate data sets for passive learning. Finally, we present our gained results. The implemented framework is available **online** [23].

3.1 Methodology

Learning setup. For automata learning, we used the learning framework AALPY [22] which is a Python library that implements many state-of-the-art automata learning algorithms. Originally, AALPY mainly focused on active automata learning algorithms. Beginning from the latest version (v.1.2.8), the tool also implements the passive learning algorithm RPNI for different modeling formalisms including Mealy machines. The latest version of AALPY is available on GitHub[1].

For active learning, we use the L^* algorithm for Mealy machines proposed by Shahbaz and Groz [33]. For conformance testing during active learning, we applied a model-based testing technique that provides state coverage. Every state is accessed n_{walk} times via the shortest prefix and then a random input sequence of length n_{step} is executed. We set $n_{\text{walks}} = 25$ and $n_{\text{step}} = 30$. For counterexample processing, we applied the improved algorithm proposed by Rivest and Schapire [31]. We used Rivest and Schapires L^* algorithm since the performed benchmark for active automata learning conducted by Aichernig et al. [4] concluded that this algorithm and the TTT algorithm [18] require approximately a similar number of queries and perform better compared to other active learning algorithms.

[1] `https://github.com/DES-Lab/AALpy`

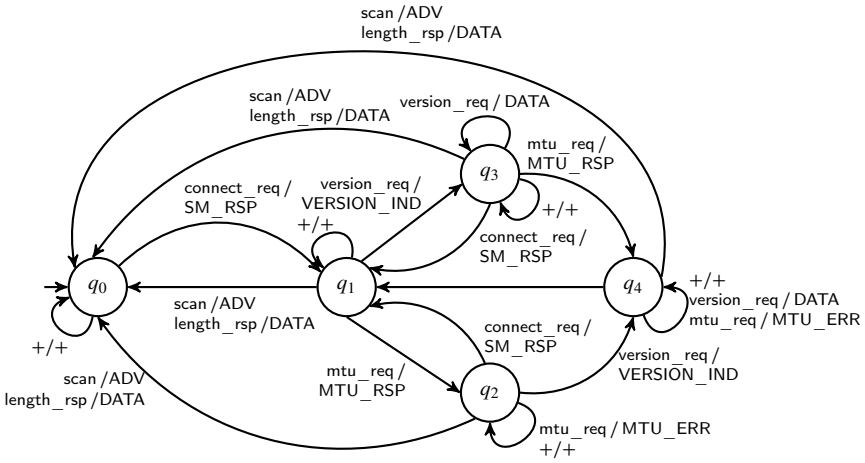


Figure 4: Model of the nRF52832. Input and output actions in the model are abbreviated or abstracted by the '+' character. The complete model and all other BLE models can be found online [23].

For passive learning, we used a variant of the RPNI algorithm for Mealy machines implemented in AALPY. To test the conformance between two Mealy machines, we require input-enabledness. Since passive data might not be complete, the passively learned model might not be input-enabled. To overcome this problem, AALPY offers two strategies for conformance testing: self-looping transitions or transitions to a sink state. For our case study, we assume the transition to a sink state in the case of an undefined input. Hence, the execution of such an undefined input always leads to a counterexample to the conformance between the actively learned and passively learned model.

We performed all experiments presented in Table 3, 4, 5, and 6 on an Apple MacBook Pro 2019 with an Intel Quad-Core i5 with 2.4 GHz and 8 GB RAM. The experiments for the heatmaps (Fig. 5) were conducted on a Dell Latitude 5410 with Intel Core i7-10610U with 2.3 GHz and 16 GB RAM.

Case study subject. In our case study, we compare active and passive learning for two network protocols: BLE and MQTT. For both protocols, we do not interact with the actual SUL but consider the already learned models from previous case studies [29, 36]. To generate output sequences, we simulate the given input sequence on the provided model, which also eases the reproducibility of our results. To maintain our black-box assumption, we do not access any further properties of the given models. Our previous work [28] introduces the BLE case study which we extended in a follow-up work [29] by learning-based fuzzing. For this evaluation, we consider the BLE devices from our learning-based fuzzing case study [29]. Figure 4 depicts the model of the BLE device *nRF52832*. We indicate the initial state by a transition with an empty source. The input and output labels are abbreviated. The model of the nRF52832 describes that during an established connection the `version_req` and `mtu_req` lead to a different output after the first execution.

For the MQTT case study, we consider a subset of the learned automata presented by Tappler et al. [36]. The MQTT automata are available as a benchmark set on the *Automata Wiki* [24]. The considered subset comprises the models of five different MQTT brokers. To evaluate larger models, the models include also the will-message functionality. All brokers interact with two clients, where one client connects with a will that the server retains. The other client can subscribe to the will's topic.

Table 2 shows the investigated examples for the BLE and MQTT case study including the number of

Table 2: The evaluation considers two case studies: BLE and MQTT. All considered automata are available online [23].

BLE			MQTT		
SUL	$\|Q\|$	$\|I\|$	*SUL*	$\|Q\|$	$\|I\|$
CC2640 [excl. feat]	11	8	ActiveMQ	18	9
CC2640 [excl. pair]	6	8	emqtt	18	9
CC2650	5	9	HBMQTT	17	9
CC2652R1	4	7	Mosquitto	18	9
CYBLE-416045-02	3	9	VerneMQ	17	9
CYW43455	4	7			
nRF52832	5	9			

states $|Q|$ and the input alphabet size $|I|$ of the ground-truth models. All considered models are available online[2] [23]. Note that for the execution on real systems, the generated traces must have been checked in advance for non-deterministic behavior. However, for passive learning, this process has to be done only once and can be separated from the learning process.

Result evaluation. For measuring the correctness of the learned models, we need to compare the learned models with the ground truth. For this, we test the conformance between the SUL and the learned model. Let $\mathscr{M}$ be a Mealy machine that represents the behavior of the SUL and $\mathscr{M}_L$ be the learned Mealy machine. We define the following conformance relation.

$$\mathscr{M}_L \textbf{ imp } \mathscr{M} \quad \Longleftrightarrow \quad \forall t \in T : \mathscr{M}_L \textbf{ passes } t. \tag{2}$$

We consider T as a finite set of traces generated from executions on $\mathscr{M}$. To generate these traces, we use two different model-based testing techniques: random and state coverage-based test generation. For both techniques, the size of the test set T is set to 10 000 traces. The generation of random traces for conformance testing is equal to the one used to generate the random sample for passive learning as later described in Algorithm 1. We define the minimum trace length $n_{\min} = 3$ and the maximum length $n_{\max} = 32$. The second model-based testing technique is based on state coverage similar to the equivalence oracle used in active learning. For this, we generate a finite set of test sequences for each state, which include the access sequence to the state and a random word as a suffix. We set the test sequences per state to $\lceil \frac{10000}{|Q|} \rceil$, where Q is the set of states of the SUL. The random word length that is executed in every state is set to 10. We provide for every technique the percentage of passed test cases. In the remaining discussion, we refer to the conformance based on state coverage, if not stated otherwise.

3.2 Data Generation

We compare active and passive automata learning based on the required data to learn the correct behavioral model of the SUL. We motivate this criterion from a theoretical and practical perspective. Theoretically, the conformance of a passively learned model to the behavior of the SUL depends on the given data set. The learned behavioral model can only cover model behavior that is included in the provided data set. Therefore, we approach the problem of finding an adequate data set that sufficiently covers the behavior of the SUL.

From the practical perspective, previous work [28, 36] showed that active automata learning could successfully create behavioral models of the actual black-box systems. However, the required active interaction and the error recovery mechanisms during learning hampered the applicability of active learning. The objective is to decrease the interaction with the SUL without missing behavioral information.

[2] `https://github.com/apferscher/ble-learning-passive`

Algorithm 1: Generation of set of random input/output traces.

Input: data set size n_{data}, minimum length n_{min}, maximum length n_{max}, SUL $\mathscr{M} = \langle Q, q_0, I, O, \delta, \lambda \rangle$
Output: $S_R \in (I \times O)^*$

```
1 S_R ⟵ [ ]
2 for i ← 1 to n_data do
3     n_len ⟵ randInt(n_min, n_max)
4     s^I ⟵ [ ]
5     for j ← 1 to n_len do
6         s^I ⟵ s^I · rand(I)
7     s^O ⟵ λ*(q_0, s^I)
8     S_R ⟵ S_R ∪ trace(s^I, s^O)
```

To tackle the theoretical as well as the practical challenge, we first evaluate if randomly generated data sufficiently represents the behavior of the SUL. Secondly, we optimize the data generated during active automata learning under the premise that the SUL's behavior is still adequately represented. Thirdly, we investigate if random sampling could support active learning.

Random Data Generation. Algorithm 1 describes the random data generation which depends on three parameters: data set size n_{data}, the minimum length of the trace n_{min}, and the maximum length of the trace n_{max}. To generate traces, we execute input sequences on Mealy machine $\mathscr{M}$ of the SUL. The algorithm returns the set S_R of generated traces. In Line 3, we select the input sequence length uniformly at random between the minimum and maximum length. Next, the algorithm selects uniformly at random an input from the input alphabet I of the SUL (Line 6). In Line 7, the input sequence is executed on the SUL to generate the corresponding output trace. Let $trace(s^I, s^O)$ be a function that creates a trace by a pairwise concatenation of inputs and outputs. In Line 8, we append the generated trace to S_R. We repeat this procedure until we generate n_{data} traces.

As a baseline for our comparison, we run Rivest and Schapire's [31] L^* algorithm variant. Let $S_{L^*} \in (I \times O)^*$ be the set of traces that L^* queries to correctly learn a Mealy machine of the SUL. Note S_{L^*} also includes the conformance tests performed during equivalence checking. We include these traces since they are required to make a statement about conformance in a black-box setup.

For the evaluation with the random data generation, we considered the following four different setups:

Sample 1: `Set of random sequences of size` $|S_{L^*}|$ `(rand` $|S_{L^*}|$`)` This experiment setup evaluates the quality of a randomly generated set with approximately the same properties as the one for active learning. The size of random sequences $n_{\text{data}} = |S_{L^*}|$. We set the minimum n_{min} and maximum n_{max} accordingly to the mean length of traces for S_{L^*}. Let $\overline{n_{L^*}}$ be the mean length of the traces in S_{L^*}, then we set $n_{\text{min}} = 1$ and $n_{\text{max}} = \lfloor 2\overline{n_{L^*}} - 1 \rceil$[3].

Sample 2: `Set of random sequences of size` $2 * |S_{L^*}|$ `(rand` $2 * |S_{L^*}|$`)` This experiment setup evaluates if a randomly generated set with approximately the same properties but doubled in size as the one for active learning is sufficient. The other parameters, n_{min} and n_{max}, are equal to **Sample 1**.

Sample 3: `Set of longer random sequences (rand long)` Given that we know the number of states of the minimal Mealy machine $\mathscr{M}$ representing the SUL, we evaluate if traces that correspond to the size of $\mathscr{M}$ are good enough. The randomly generated traces are at least $|Q|$ long, $n_{\text{min}} = |Q|$, and have a maximum length of $n_{\text{max}} = 2 * |Q|$. The random set is of size $|S_{L^*}|$.

Sample 4: `Set of sufficient random samples (rand corr)` The goal is to find a parameter setup

[3] $\lfloor x \rceil$ denotes the nearest natural number to the real number x, where a distance of 0.5 is associate to the higher natural number.

that generates a random sample that sufficiently represents the SUL. For this, we evaluated how the trace length and the data set size influence the success of learning. Based on these two parameters we generate different random samples.

Optimized Data Generation. Our second data set approaches the minimization of the data set generated by the L^* algorithm. Due to the design of the used version of the L^* algorithm, we assume that the generated data set is not minimal. The data set generated by the L^* algorithm is not minimal due to two aspects: performed queries might be a prefix of other queries and a non-minimal characterization set. Firstly, considering the incremental state exploration of the L^* algorithm, we assume that some traces are a prefix of other traces that are later queried in the L^* algorithm. Since the RPNI algorithm relies on the construction of a PTA, traces that are a prefix of another trace do not provide any further knowledge for passive learning. Secondly, we can make a reduction of the required traces based on the characterization set of the learned Mealy machine. The characterization set is a set of input sequences that generate different output sequences for each state of the SUL. In the L^* algorithm version proposed by Shahbaz and Groz [33], introduced in Sect. 2, the E set defines the characterization set and is initialized with the whole input alphabet. However, not necessarily all inputs are required to distinguish a state. Consider the observation table, Table 1, presented in Sect. 2. In this table, the input i_2 does not provide any information that enables the distinction of states. Hence, i_2 is not part of the characterization set and is not required in set E. We, therefore, can remove all queries that are required to fill this column. To get an optimized data set, we calculate the characterization set based on the Mealy machine of the SUL. The reduced data set is then used as a base for passive learning and we expect the learned automaton to be equal to the Mealy machine of the SUL.

Cache in active learning. Our third evaluation investigates if random data could support active automata learning. The learning library AALPY supports the caching of queries using a PTA. Before executing a query, the algorithm first checks if the query is already included in the PTA. If the PTA includes the query output, then the query is not performed on the SUL. This reduces the number of required interactions with the SUL. In our evaluation, we investigate the impact on the required SUL interactions with a cache initialized by random samples. For this, we initialize the cache of the L^* algorithm with a random sample that has the same properties as the one initially required by L^*, i.e. **Sample 1**.

3.3 Findings

Table 3 and 4 present the results of actively learning the BLE and MQTT models using the L^* algorithm. All learned models conform to the original models of the corresponding SUL. In addition, the tables show the minimized data size, where the minimization is performed as described in Sect. 3.2.

A comparison of both tables shows that the MQTT example requires significantly more queries to learn the final model. We explain this increase based on two observations. Firstly, Table 2 shows that the state space for the MQTT examples is larger. Secondly, the MQTT examples require more than one learning round. This circumstance substantially increases the sum of performed queries during active learning. All BLE models could be learned within one learning round. Hence, the conformance test during learning was only performed once and did not add any new behavior to the SUL. However, the active learning of the MQTT examples requires at least three conformance testing rounds. Regarding the minimization of data, we see a significant decrease in the data that is actually required to learn correctly. This observation agrees with our assumption that the data set includes redundant and non-essential queries.

Table 5 and Table 6 present the evaluation results of passive learning considering the different randomly generated data sets **Sample 1-3**. For **Sample 4**, we created a heatmap which is presented in Fig. 5.

Table 3: Active learning results for the BLE case study. Every experiment was repeated five times. Every device could be learned within one learning round. Hence, the reported results were deterministic.

	CC2640 excl. feat.	**CC2640 excl. pair.**	**CC2650**	**CC2652R1**	**CYBLE-416045-02**	**CYW43455**	**nRF52832**
Output queries	704	384	405	196	243	784	405
Output queries steps	3136	1472	1458	588	729	3136	1458
Conformance tests	275	150	125	100	75	400	125
Conformance tests steps	8925	4775	3950	3100	2325	12800	3950
Sum queries	979	534	530	296	318	1184	530
Sum steps	12061	6247	5408	3688	3054	15936	5408
$\overline{n_{len}}$	12.32	11.70	10.20	12.46	9.60	13.46	10.20
Optimized queries	312	129	123	50	50	388	123
Optimized $\overline{n_{len}}$	4.55	3.9	3.65	3.08	3.04	4.13	3.65

Table 4: Active learning results for the MQTT case study. Every experiment was repeated five times, numbers in brackets show the standard deviation. If no standard deviation is given, the deviation is zero.

	ActiveMQ	**emqtt**	**HBMQTT**	**Mosquitto**	**VerneMQ**
Output queries	5890.40 (695.77)	5900.60 (1834.84)	4009.40 (533.83)	4056.40 (846.08)	4356.20 (976.87)
Output queries steps	55771.60 (14265.88)	62134.80 (33276.46)	33176.40 (8277.56)	34335.80 (10979.61)	33850.80 (12344.85)
Conformance tests	450	450	425	450	425
Conformance tests steps	14721.2 (70.33)	14654.2 (24.69)	13794 (16.41)	14737 (88.12)	13834.4 (87.33)
Learning rounds	5.20 (1.30)	5.20 (1.30)	3.60 (0.55)	4.00 (1.00)	5.20 (1.10)
Sum queries	6340.40 (695.77)	6350.60 (1834.84)	4434.40 (533.83)	4506.40 (846.08)	4781.20 (976.87)
Sum steps	70492.80 (14208.67)	76789.00 (33275.29)	46970.40 (8282.37)	49072.80 (10998.34)	47685.20 (12364.07)
$\overline{n_{len}}$	11.04 (1.18)	11.77 (2.21)	10.55 (0.86)	10.85 (1.00)	9.90 (0.66)
Optimized queries	1450	1450	959	1015	959
Optimized $\overline{n_{len}}$	6.26	6.26	5.91	6.05	6.05

For each experiment setup, we state the size of the learning set n_{data}, the mean length of the traces $\overline{n_{\text{len}}}$, the conformance to the SUL, and the number of correctly learned models. Every experiment was repeated five times and we give the mean value of these five repetitions. We consider here one exception: the number of correctly learned models gives the absolute number and not the mean. For example, if we write "1" then one out of five learning repetitions learned the correct automaton.

In the following, we describe the obtained results based on our three research questions.

(RQ 1) *Can passive learning based on a random sample outperform active learning?* A comparison of Table 5 and Table 6 shows that the obtained results for two case studies provide a slightly different conclusion. For the BLE case study, in some cases, it was possible to learn the correct model with a random sample size equal to the active sample size. More precisely, for one BLE device, CYW43455, we always learn correctly with a random sample. We could also correctly learn for most experiments of the CC2652R1. The CYW43455 and CC2652R1 are two SULs with a smaller input alphabet size. We see that if the number of inputs increases, the chance to learn correctly decreases. Note that for the BLE case study the average length of **Sample 3** is shorter than the one of **Sample 1**. This is due to the fact that the generation of **Sample 3** is based on the size of the SUL.

For the MQTT case study (Table 6), we never managed to learn correctly with the considered random samples. Even if active learning requires a large set of queries, passive learning seems to be no alternative for this case study. However, the passive learning conformance was always quite high with around 99% conformance. A closer look at the passively learned automata revealed that the size of the learned automata was considerably larger than the size of the ground truth automata. For example, the average

Table 5: Passive automata learning results for the BLE case study. Every experiment was repeated five times. The table gives the mean value and the corresponding standard deviation in brackets. For the number of correctly learned models, the sum over all five experiments is given.

		CC2640 excl. feat.	CC2640 excl. pair.	CC2650	CC2652R1	CYBLE-416045-02	CYW43455	nRF52832
Sample 1 `rand` $\lvert S_{L^*}\rvert$	n_{data}	979.00	534.00	530.00	296.00	318.00	1184.00	530.00
	$\overline{n_{\text{len}}}$	12.64 (0.25)	11.46 (0.32)	9.84 (0.30)	12.69 (0.37)	9.47 (0.23)	13.54 (0.20)	10.04 (0.24)
	Conformance (random) %	99.92 (0.01)	99.93 (0.02)	99.89 (0.06)	100.00 (0.00)	99.93 (0.06)	100.00 (0.00)	99.94 (0.03)
	Conformance (coverage) %	99.80 (0.02)	99.86 (0.03)	99.82 (0.08)	100.00 (0.00)	99.89 (0.08)	100.00 (0.00)	99.85 (0.09)
	Correct model	0	0	0	5	1	5	0
Sample 2 `rand` $2 * \lvert S_{L^*}\rvert$	n_{data}	1958.00 (0.00)	1068.00 (0.00)	1060.00 (0.00)	592.00 (0.00)	636.00 (0.00)	2368.00 (0.00)	1060.00 (0.00)
	$\overline{n_{\text{len}}}$	12.49 (0.16)	11.62 (0.15)	10.02 (0.20)	12.33 (0.24)	9.61 (0.16)	13.55 (0.10)	10.10 (0.15)
	Conformance (random) %	99.97 (0.01)	99.98 (0.03)	99.98 (0.02)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	99.97 (0.02)
	Conformance (coverage) %	99.92 (0.04)	99.96 (0.06)	99.96 (0.04)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	99.93 (0.06)
	Correct model	0	2	2	5	5	5	0
Sample 3 `rand` `long`	n_{data}	979.00	534.00	530.00	296.00	318.00	1184.00	530.00
	$\overline{n_{\text{len}}}$	16.55 (0.05)	9.03 (0.07)	7.46 (0.09)	5.99 (0.07)	4.49 (0.10)	24.08 (0.12)	7.54 (0.07)
	Conformance (random) %	99.93 (0.04)	99.87 (0.04)	99.87 (0.06)	99.81 (0.14)	99.79 (0.10)	100.00 (0.00)	99.91 (0.03)
	Conformance (coverage) %	99.83 (0.09)	99.76 (0.07)	99.76 (0.08)	99.86 (0.10)	99.72 (0.11)	100.00 (0.00)	99.81 (0.04)
	Correct model	0	0	0	1	0	5	0

model size for HBMQTT based on **Sample 1** was 64.4 instead of 17. We assume that this increase in size results from sparse data. Since observations are missing, states could not be merged.

For one BLE and MQTT device, we evaluated how the average trace length and the number of samples influence the success of passive learning (**Sample 4**). Figure 5 presents the results in two heatmaps. The x-axis indicates a factor by which the L^* learning sample size $|S_{L^*}|$ is multiplied. The y-axis gives the mean length of the traces. The darker the green and the higher the number, the more the passively learned model conforms to the SUL. We observe that shorter traces are not sufficient to learn correctly, even if the sample size is large. However, to achieve 100% conformance large random samples with long traces are required. Hence, considering that the interaction with the SUL should be decreased, passive learning does not provide an alternative.

(RQ 2) *Does the considered active automata learning algorithm generate an optimal sample?* Since passive learning does not improve the number of required interactions with the SUL, we investigate if there still is potential for optimization in active learning. To evaluate the potential of active learning, we generated an optimized learning sample set as described in Sect. 3.2. With this optimized sample, passive learning could correctly learn the minimal model of the SUL for all examples. Table 3 and 4 show the same potential for improvement in both case studies. On average, the BLE active learning sample size could be reduced by 76% and for MQTT by 78%. For example, the BLE device CC2652R1 could be learned with a sample of size 50 instead of 296. Additionally, the average length can be significantly reduced. The reduction for BLE is on average 67% and for MQTT 43%.

If we compare the experimental results between the random data sizes shown in Fig. 5 and the learning with the data set from L^*, we conclude that the generation of a sufficient data set via random

Table 6: Passive automata learning results for the MQTT case study. The data presentation is similar to Table 5. The values in the brackets indicate the standard deviation.

		ActiveMQ	emqtt	HBMQTT	Mosquitto	VerneMQ
Sample 1 `rand` $\lvert S_{L^*}\rvert$	n_{data}	6340.00	6351.00	4434.00	4506.00	4781.00
	$\overline{n_{\text{len}}}$	11.00	12.00	10.46	10.98	10.07
		(0.11)	(0.05)	(0.08)	(0.06)	(0.10)
	Conformance	99.97	99.96	99.85	99.95	99.95
	(random) %	(0.01)	(0.01)	(0.01)	(0.00)	(0.02)
	Conformance	99.95	99.94	99.79	99.91	99.92
	(coverage) %	(0.02)	(0.02)	(0.02)	(0.01)	(0.04)
	Correct model	0	0	0	0	0
Sample 2 `rand` $2 * \lvert S_{L^*}\rvert$	n_{data}	12680.00	12702.00	8868.00	9012.00	9562.00
	$\overline{n_{\text{len}}}$	11.02	11.99	10.54	11.00	9.94
		(0.06)	(0.03)	(0.03)	(0.04)	(0.05)
	Conformance	99.98	99.97	99.98	99.98	99.98
	(random) %	(0.01)	(0.01)	(0.01)	(0.01)	(0.01)
	Conformance	99.97	99.97	99.96	99.96	99.97
	(coverage) %	(0.01)	(0.01)	(0.02)	(0.02)	(0.02)
	Correct model	0	0	0	0	0
Sample 3 `rand` `long`	n_{data}	6340.00	6351.00	4434.00	4506.00	4781.00
	$\overline{n_{\text{len}}}$	26.99	27.00	25.52	27.00	25.45
		(0.05)	(0.02)	(0.02)	(0.15)	(0.12)
	Conformance	99.97	99.97	99.99	99.95	99.97
	(random) %	(0.01)	(0.01)	(0.01)	(0.06)	(0.02)
	Conformance	99.96	99.96	99.97	99.94	99.97
	(coverage) %	(0.02)	(0.02)	(0.01)	(0.06)	(0.02)
	Correct model	0	0	0	0	0

techniques requires significantly more traces than the L^* data set. For example, the heatmap on the right side representing the results of the MQTT broker 'Mosquitto' shows that even if the random data set size is 9 times larger and the average trace length is approximately 9 inputs longer than in the set required by L^*, passive learning still is not able to learn correctly. These results motivate the usage of active automata learning techniques. Based on the observation of Aichernig et al. [4], the choice of the learning algorithm and the corresponding conformance testing technique influences the required interaction with the SUL. Further improvements might be observable by the usage of other active learning algorithms, e.g. TTT [18], that avoid redundancy in the performed queries.

(RQ 3) *Can random sampling support active automata learning?*

To reduce the redundancy of performed queries, we supported active automata learning by the usage of a cache. The motivation is to avoid query executions on the SUL since the observations can be retrieved from data already visible in the cache. For this, we initialized the cache with a random sample as it is generated in **Sample 1**. Hence, the size of the sample conforms to the number of queries required by the L^* algorithm to learn correctly. In the performed experiments, we measure how many additional queries are required by the L^* algorithm to generate a conforming model.

Our obtained results show that a cache initialized by a random sample only covers the minority of the data that must be queried by active learning. On average, for the BLE case study, randomly generated data only represents 27% of the total data required to learn correctly. As a result, additionally 63% more queries had to be made by active learning. For MQTT, only 10.6% of the performed queries were already present in the randomly initialized cache.

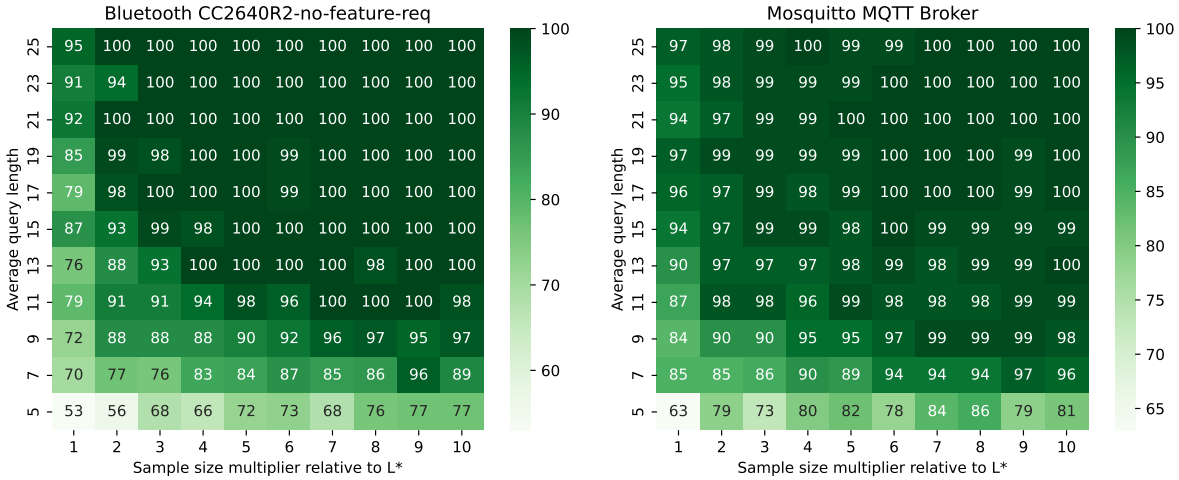


Figure 5: Each heatmap presents, for one example of the BLE and MQTT case study, the influence of size and average trace length required to learn correctly.

4 Related Work

In the literature, the topic of model inference from network protocols is well studied. Several presented approaches used active automata learning to generate behavioral models of different protocols like 802.11 4-way handshake [35], BLE [28, 29], (D)TLS [32, 13], MQTT [36], QUIC [30], SSH [14], or TCP [12]. They motivate the usage of an active technique by considering it as a testing approach. Assuming that the system is input-enabled, every input is executed in every state. Therefore, input sequences are tested that might be rare during common usage. This technique is also known as protocol state fuzzing since the execution of unusual input sequences might reveal unexpected behavior.

There is also work that uses passive techniques. Comparetti et al. [10] presented a general framework for passively learning communication protocols. Their evaluation includes protocols like malware bots, SMTP, or SIP. For learning, they used recorded sessions. Hence, their learned behavioral models only cover parts of the protocol that can be observed during normal user sessions. Other behavioral aspects have been excluded. Doupé et al. [11] generate behavioral models of web applications. Their proposed technique uses a web crawler to generate a set of traces that navigates through the web application. These crawled traces are then used to infer a model of the web application. Both passive techniques [10, 11] use their inferred models to fuzz test the SUL.

The comparison of automata learning techniques mainly focuses on algorithms of the same paradigm. Initially, the Zulu challenge [9] addressed the problem of performing equivalence queries in active automata learning. The goal is to establish a conformance testing technique that can effectively test equivalence considering a limited testing budget. This challenge has now been replaced by the Rigorous Examination of Reactive Systems (RERS) [19] challenge. RERS provides new benchmarks for evaluating learning techniques based on reactive systems. Aichernig et al. [4] published a comprehensive evaluation of different combinations of active automata learning algorithms and conformance testing techniques. In their work, they conclude that the improved L^* version from Rivest and Schapire [31] or the TTT algorithm [18] should be used for active learning. For conformance testing, they recommend the randomized version of the partial W-method [15] or a mutation-based technique [3]. For passive learning, Lang et al. [20] proposed a challenge to passively learn DFAs via state merging, including also

the aspect of sparse data. The winning algorithm is based on evidence-driven state merging which was able to generalize better on sparse data. Lo et al. [21] performed an empirical case study that evaluated different passive learning algorithms, e.g. kTail [6], also considering sparse data. A comparison of passive and active learning was performed by Aichernig et al. [2]. In their work, they compared a passive and an active variant of a search-based learning algorithm for timed automata. Their comparison shows that the active algorithm outperforms the passive variant in terms of required data to learn correctly.

5 Conclusion

Summary. Active automata learning of network protocols has been an active research area in recent years. However, this technique might be limited in practice since a fault-tolerant learning setup that enables active interaction is required. This might hamper the application to autonomous systems. Passive automata learning, instead, infers a behavioral model from a given data set and does not require an interface that enables an active interaction with the SUL. Motivated by this, we compared active and passive learning by evaluating practical case studies. Our results show that we require a larger randomly generated sample for passive learning than the data set used by active automata learning. Furthermore, we showed that the data set generated by active learning can be optimized in terms of size. In summary, our results justify the costs of establishing an interface that allows an active interaction with the SUL.

Discussion. The primary goal for the inference of network protocols is to learn a conforming model with the lowest possible number of interactions. This goal is motivated by the fact that the interaction over a network to the SUL can be expensive in the sense that queries might be repeated since they were lost or arrived delayed. Our presented evaluation discusses if passive learning paradigms can overcome this problem of active interaction since they are based on a given data set. In practice, log files might represent such a data set. However, for our presented evaluation, we considered a randomly generated data set, since real-world logging data is assumed to be incomplete. For example, consider the BLE use case: logging data would probably not contain any parameter request that is not required for the establishment of a valid connection. The comparison with L^* shows that even an active learning algorithm that queries a lot of redundant information is better than passive learning. However, our experiments on optimizing the data set of the L^* algorithm indicate that there is room for improvements also on the side of active learning. We expect that other active algorithms, like TTT [18], or other conformance testing techniques can help to improve active learning in the reduction of the required interaction with the SUL.

Future Work. For future work, it would be interesting to compare active and passive automata learning based on other practical case studies discussed in the literature. For example, to consider also additionally modeling formalisms like stochastic or timed systems, for which also active and passive learning algorithms exist. Additionally, a general comparison between several active and passive learning techniques on theoretical challenges, like RERS [19], could be useful. Furthermore, other learning algorithms should be considered to further optimize the required data set. A different direction would be the evaluation of a combination of passive and active learning as proposed by Walkinshaw et al. [38]. Such an approach might unite the advantages of both learning paradigms since the proposed technique only requires an active interaction for conformance testing. Such a technique might reduce the required interaction with an SUL compared to an entirely active technique.

Acknowledgement. This work was done in the TU Graz LEAD project “Dependable Internet of Things in Adverse Environments”, the LearnTwins project funded by FFG (Österreichische Forschungsförderungsgesellschaft) under grant 880852, and the “University SAL Labs” initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

References

[1] Bernhard K. Aichernig, Edi Muškardin & Andrea Pferscher (2021): *Learning-Based Fuzzing of IoT Message Brokers*. In: *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12-16, 2021*, IEEE, pp. 47–58, doi:10.1109/ICST49551.2021.00017.

[2] Bernhard K. Aichernig, Andrea Pferscher & Martin Tappler (2020): *From Passive to Active: Learning Timed Automata Efficiently*. In Ritchie Lee, Susmit Jha & Anastasia Mavridou, editors: *NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11-15, 2020, Proceedings*, *Lecture Notes in Computer Science* 12229, Springer, pp. 1–19, doi:10.1007/978-3-030-55754-6_1.

[3] Bernhard K. Aichernig & Martin Tappler (2019): *Efficient Active Automata Learning via Mutation Testing*. *J. Autom. Reason.* 63(4), pp. 1103–1134, doi:10.1007/s10817-018-9486-0.

[4] Bernhard K. Aichernig, Martin Tappler & Felix Wallner (2020): *Benchmarking Combinations of Learning and Testing Algorithms for Active Automata Learning*. In Wolfgang Ahrendt & Heike Wehrheim, editors: *Tests and Proofs - 14th International Conference, TAP@STAF 2020, Bergen, Norway, June 22-23, 2020, Proceedings*, *Lecture Notes in Computer Science* 12165, Springer, pp. 3–22, doi:10.1007/978-3-030-50995-8_1.

[5] Dana Angluin (1987): *Learning Regular Sets from Queries and Counterexamples*. *Inf. Comput.* 75(2), pp. 87–106, doi:10.1016/0890-5401(87)90052-6.

[6] Alan W. Biermann & Jerome A. Feldman (1972): *On the Synthesis of Finite-State Machines from Samples of Their Behavior*. *IEEE Trans. Computers* 21(6), pp. 592–597, doi:10.1109/TC.1972.5009015.

[7] Bluetooth SIG (2021): *Bluetooth Core Specification v5.3*. Standard, Bluetooth SIG. Available at `https://www.bluetooth.com/specifications/specs/core-specification/`.

[8] Ana Cavalcanti & Dennis Dams, editors (2009): *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*. *Lecture Notes in Computer Science* 5850, Springer, doi:10.1007/978-3-642-05089-3.

[9] David Combe, Colin de la Higuera & Jean-Christophe Janodet (2009): *Zulu: An Interactive Learning Competition*. In Anssi Yli-Jyrä, András Kornai, Jacques Sakarovitch & Bruce W. Watson, editors: *Finite-State Methods and Natural Language Processing, 8th International Workshop, FSMNLP 2009, Pretoria, South Africa, July 21-24, 2009, Revised Selected Papers*, *Lecture Notes in Computer Science* 6062, Springer, pp. 139–146, doi:10.1007/978-3-642-14684-8_15.

[10] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Krügel & Engin Kirda (2009): *Prospex: Protocol Specification Extraction*. In: *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, IEEE Computer Society, pp. 110–125, doi:10.1109/SP.2009.14.

[11] Adam Doupé, Ludovico Cavedon, Christopher Kruegel & Giovanni Vigna (2012): *Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner*. In Tadayoshi Kohno, editor: *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, USENIX Association, pp. 523–538. Available at `https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/doupe`.

[12] Paul Fiterau-Brostean, Ramon Janssen & Frits W. Vaandrager (2016): *Combining Model Learning and Model Checking to Analyze TCP Implementations*. In Swarat Chaudhuri & Azadeh Farzan, editors: *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, *Lecture Notes in Computer Science* 9780, Springer, pp. 454–471, doi:10.1007/978-3-319-41540-6_25.

[13] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas & Juraj Somorovsky (2020): *Analysis of DTLS Implementations Using Protocol State Fuzzing*. In Srdjan Capkun & Franziska Roesner, editors: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, USENIX Association, pp. 2523–2540. Available at `https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brostean`.

[14] Paul Fiterau-Brostean, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits W. Vaandrager & Patrick Verleg (2017): *Model learning and model checking of SSH implementations*. In Hakan Erdogmus & Klaus Havelund, editors: *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, ACM, pp. 142–151, doi:10.1145/3092282.3092289.

[15] Susumu Fujiwara, Gregor von Bochmann, Ferhat Khendek, Mokhtar Amalou & Abderrazak Ghedamsi (1991): *Test Selection Based on Finite State Models*. *IEEE Trans. Software Eng.* 17(6), pp. 591–603, doi:10.1109/32.87284.

[16] E. Mark Gold (1978): *Complexity of Automaton Identification from Given Data*. *Inf. Control.* 37(3), pp. 302–320, doi:10.1016/S0019-9958(78)90562-4.

[17] Colin de la Higuera (2010): *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9781139194655.

[18] Malte Isberner, Falk Howar & Bernhard Steffen (2014): *The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning*. In Borzoo Bonakdarpour & Scott A. Smolka, editors: *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, *Lecture Notes in Computer Science* 8734, Springer, pp. 307–322, doi:10.1007/978-3-319-11164-3_26.

[19] Marc Jasper, Maximilian Fecke, Bernhard Steffen, Markus Schordan, Jeroen Meijer, Jaco van de Pol, Falk Howar & Stephen F. Siegel (2017): *The RERS 2017 challenge and workshop (invited paper)*. In Hakan Erdogmus & Klaus Havelund, editors: *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, ACM, pp. 11–20, doi:10.1145/3092282.3098206.

[20] Kevin J. Lang, Barak A. Pearlmutter & Rodney A. Price (1998): *Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm*. In Vasant G. Honavar & Giora Slutzki, editors: *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, *Lecture Notes in Computer Science* 1433, Springer, pp. 1–12, doi:10.1007/BFb0054059.

[21] David Lo, Leonardo Mariani & Mauro Santoro (2012): *Learning extended FSA from software: An empirical assessment*. *J. Syst. Softw.* 85(9), pp. 2063–2076, doi:10.1016/j.jss.2012.04.001.

[22] Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher & Martin Tappler (2022): *AALpy: An Active Automata Learning Library*. *Innovations Syst. Softw. Eng.*, doi:10.1007/s11334-022-00449-3.

[23] Edi Muškardin & Andrea Pferscher: *Supplemental Material for "Active vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols"*. `https://github.com/apferscher/ble-learning-passive`. Accessed: 2022-08-05.

[24] Daniel Neider, Rick Smetsers, Frits W. Vaandrager & Harco Kuppens (2018): *Benchmarks for Automata Learning and Conformance Testing*. In Tiziana Margaria, Susanne Graf & Kim G. Larsen, editors: *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science* 11200, Springer, pp. 390–416, doi:10.1007/978-3-030-22348-9_23.

[25] OASIS: *MQTT Version 5.0*. `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf`. Acessed: 2022-08-04.

[26] José Oncina & Pedro Garía (1993): *Identifying regular languages in polynomial time*. *Advances in Structural and Syntactic Pattern Recognition* 5, pp. 99–108, doi:10.1142/9789812797919_0007.

[27] Andrea Pferscher & Bernhard K. Aichernig (2020): *Learning Abstracted Non-deterministic Finite State Machines*. In Valentina Casola, Alessandra De Benedictis & Massimiliano Rak, editors: *Testing Software and Systems - 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9-11, 2020, Proceedings*, *Lecture Notes in Computer Science* 12543, Springer, pp. 52–69, doi:10.1007/978-3-030-64881-7_4.

[28] Andrea Pferscher & Bernhard K. Aichernig (2021): *Fingerprinting Bluetooth Low Energy Devices via Active Automata Learning*. In Marieke Huisman, Corina S. Pasareanu & Naijun Zhan, editors: *Formal Methods -*

24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings, *Lecture Notes in Computer Science* 13047, Springer, pp. 524–542, doi:10.1007/978-3-030-90870-6_28.

[29] Andrea Pferscher & Bernhard K. Aichernig (2022): *Stateful Black-Box Fuzzing of Bluetooth Devices Using Automata Learning*. In Jyotirmoy V. Deshmukh, Klaus Havelund & Ivan Perez, editors: *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, *Lecture Notes in Computer Science* 13260, Springer, pp. 373–392, doi:10.1007/978-3-031-06773-0_20.

[30] Abdullah Rasool, Greg Alpár & Joeri de Ruiter (2019): *State machine inference of QUIC*. *CoRR* abs/1903.04384, doi:10.48550/arXiv.1903.04384.

[31] Ronald L. Rivest & Robert E. Schapire (1993): *Inference of Finite Automata Using Homing Sequences*. *Inf. Comput.* 103(2), pp. 299–347, doi:10.1006/inco.1993.1021.

[32] Joeri de Ruiter & Erik Poll (2015): *Protocol State Fuzzing of TLS Implementations*. In Jaeyeon Jung & Thorsten Holz, editors: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, USENIX Association, pp. 193–206. Available at `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter`.

[33] Muzammil Shahbaz & Roland Groz (2009): *Inferring Mealy Machines*. In Cavalcanti & Dams [8], pp. 207–222, doi:10.1007/978-3-642-05089-3_14.

[34] Wouter Smeenk, Joshua Moerman, Frits W. Vaandrager & David N. Jansen (2015): *Applying Automata Learning to Embedded Control Software*. In Michael J. Butler, Sylvain Conchon & Fatiha Zaïdi, editors: *Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings*, *Lecture Notes in Computer Science* 9407, Springer, pp. 67–83, doi:10.1007/978-3-319-25423-4_5.

[35] Chris McMahon Stone, Tom Chothia & Joeri de Ruiter (2018): *Extending Automated Protocol State Learning for the 802.11 4-Way Handshake*. In Javier López, Jianying Zhou & Miguel Soriano, editors: *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, *Lecture Notes in Computer Science* 11098, Springer, pp. 325–345, doi:10.1007/978-3-319-99073-6_16.

[36] Martin Tappler, Bernhard K. Aichernig & Roderick Bloem (2017): *Model-Based Testing IoT Communication via Active Automata Learning*. In: *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, IEEE Computer Society, pp. 276–287, doi:10.1109/ICST.2017.32.

[37] Jan Tretmans (2008): *Model Based Testing with Labelled Transition Systems*. In Robert M. Hierons, Jonathan P. Bowen & Mark Harman, editors: *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, *Lecture Notes in Computer Science* 4949, Springer, pp. 1–38, doi:10.1007/978-3-540-78917-8_1.

[38] Neil Walkinshaw, John Derrick & Qiang Guo (2009): *Iterative Refinement of Reverse-Engineered Models by Model-Based Testing*. In Cavalcanti & Dams [8], pp. 305–320, doi:10.1007/978-3-642-05089-3_20.