Evaluating Impact of Coverage Feedback on Estimators for Maximum Reachability in Fuzzing (Registered Report)

Nelum Attanayake, Danushka Liyanage, Clément Canonne, Suranga Seneviratne, and Rahul Gopinath
School of Computer Science, University of Sydney
{nelum.attanayake, danushka.liyanage, clement.canonne, suranga.seneviratne, rahul.gopinath}@sydney.edu.au

***Abstract*—Background: Fuzzing campaigns require accurate estimation of maximum reachable coverage to ensure that resources are not wasted. However, adaptive bias due to the use of coverage feedback in modern fuzzers prevents accurate statistical estimation of maximum reachable coverage. Recent work hypothesizes that adaptive bias is minimized when singleton species, observed exactly once, equal doubletons, observed exactly twice. Rigorous evaluation of this hypothesis has been hindered by the lack of ground truth.**

Objective: This work evaluates whether maximum reachable coverage estimates are reliable when adaptive bias is minimized, using two complementary approaches (1) to mitigate the lack of ground truth and (2) to establish ground truth.

Methods: First, we compare maximum reachable coverage estimates between coverage-guided and purely random fuzzers on real-world benchmarks. Since random fuzzers lack coverage feedback, they exhibit no adaptive bias. If the singleton-doubleton equilibrium criterion reliably indicates minimal adaptive bias, the coverage-guided fuzzer should reach maximum reachable coverage estimates comparable to the random fuzzer at this equilibrium point. Second, we validate estimates using synthetic programs with known maximum reachable coverage, where complex control flows mimic real-world complexity while providing objective ground truth.

Results: These complementary studies will determine whether maximum reachable coverage estimates are reliable when the singleton-doubleton equilibrium criterion is satisfied, validating or refuting its use as a stopping criterion for fuzzing campaigns.

I. Introduction

Greybox fuzzing is an automated testing technique that leverages coverage feedback [1] to find bugs and vulnerabilities. To find bugs, fuzzers must generate inputs that *cover* faulty code. The primary goal, therefore, is to maximize coverage. However, 100% coverage is nearly impossible to achieve in real-world programs as many of the program elements are unreachable under any input [2], [3].

As fuzzing campaigns are computation and resource intensive, they should only proceed until the fuzzer has reached *coverage saturation* of the subject. Yet, determining this *maximum reachable coverage* beyond which *further fuzzing will not produce more coverage* is an open problem in real-world programs [4]. Hence, a key question is: *when to stop*.

Statistical methods are widely used for estimating maximum reachable coverage. The best known are the *species richness estimators* from biostatistics [5], which treats fuzzing as a statistical sampling process where each input exercises a specific program element, which can be used to estimate the number of such program elements from the covered elements [6]. In this analogy, each distinct program element is treated as a *species*, and statistical properties of the observed frequencies inform estimates of the total number of elements.

Prior evaluations [7] of the species richness estimators have shown that existing estimators tend to *systematically underestimate* the true maximum reachable coverage. A potential reason for this bias is the *adaptive heuristics* inherent in greybox fuzzing due to the use of coverage feedback. That is, more program elements are covered faster than one would expect from a purely random fuzzer with no information about the coverage achieved by previous inputs.

One way to mitigate this issue is to wait until the utility of coverage feedback is minimal, which occurs late in the fuzzing campaign [8]. Researchers have recently [9] claimed that the coverage feedback has been adequately exhausted when the number of program elements covered exactly once, f_1 (i.e., singleton species), equals the number of program elements covered exactly twice, f_2 (i.e., doubleton species). This is called the *singleton-doubleton equilibrium* (referred to as the $f_1 f_2$ equilibrium from now on). At this point, estimates produced from species richness estimators should be reliable. If this claim is accurate, then it provides a reasonable *stopping criterion* for greybox fuzzing: fuzz until the $f_1 f_2$ equilibrium is reached, produce the maximum reachable coverage estimate, and continue fuzzing until that estimated coverage is attained. Such estimates can also inform residual defect estimates when combined with domain-specific defect density assumptions.

In this work, we investigate the validity of this claim from two complementary perspectives. First, we examine whether coverage feedback is indeed the reason for underestimation of the maximum reachable coverage, and whether its influence is sufficiently minimized at the $f_1 f_2$ equilibrium to produce reliable estimates. To test this hypothesis, we compare the maximum reachable coverage estimated with

Fuzzing Workshop (FUZZING) 2026
27 February 2026, San Diego, CA, USA
ISBN 978-1-970672-04-6
https://dx.doi.org/10.14722/fuzzing.2026.23010
www.ndss-symposium.org

non-directed blackbox fuzzing[1] on real-world programs, with that obtained using feedback-directed greybox fuzzing. Since blackbox fuzzing lacks coverage feedback and thus has no adaptive bias [8], we hypothesize that the two estimates should converge when adaptive bias in greybox fuzzing is minimal. We note that it is possible to construct programs where $f_1 f_2$ equilibrium does not exist. However, the STADS framework [6] implicitly assumes that such instances are rare, and programs can be treated statistically in general. Hence, one of the aims of this experiment is to validate whether this assumption is true. Hence, the first research question is:

RQ1: Does the $f_1 f_2$ equilibrium coincide with overlapping estimates from pure blackbox and greybox fuzzing? The result of this experiment will validate or refute the impact of adaptive bias on maximum reachable coverage estimate.

However, comparing fuzzer types alone cannot definitively validate the accuracy of maximum reachable coverage estimates without ground truth. To address this limitation, we adopt a complementary evaluation strategy. Recent research suggests that *recursive descent parsers* generated from arbitrary context-free LL(1) grammars can mimic the control-flow complexity of real-world programs while providing ground truth [11]. We will use such arbitrarily generated recursive descent parsers as synthetic subjects with known maximum reachable coverage. Hence, the second research question is:

RQ2: How accurate are the maximum reachable coverage estimators compared to ground truth? The result of this experiment will validate the estimators at minimal adaptive bias.

Contribution: This report evaluates the reliability of maximum reachable coverage estimates at $f_1 f_2$ equilibrium from two complementary experiments:

- An empirical evaluation of the hypothesis that at $f_1 f_2$ equilibrium, maximum reachable coverage estimates from greybox and blackbox fuzzers coincide with real-world programs.
- An empirical evaluation of the claim that when maximum reachable coverage estimate is reliable at the $f_1 f_2$ equilibrium with synthetic programs.

The remainder of this report is organized as follows. We begin with the *preliminaries* in Section II, providing the necessary background on fuzzing, statistical estimators, and benchmark generation. This is followed by the *execution plan* in Section III, which details our experimental setup, data collection, and analysis methodology. Section IV then explains the *evaluation metrics* used in our analysis. Next, we review the *related work* in Section V to situate our study within the broader context of fuzzing research, followed by a discussion of *threats to validity* in Section VI. We then discuss *future work* in Section VII before concluding with a summary of our findings and key takeaways in Section VIII.

II. Preliminaries

In this section, we present the statistical model underlying our effectiveness estimators, along with the stopping criterion used to evaluate campaign progression and termination.

A. Probabilistic Model for Fuzzing

The STADS[2] framework [6], [8], [12], [13] models fuzzing as a statistical sampling process $\mathcal{F}$. Each test input is drawn with replacement from the program's input space $\boldsymbol{\mathcal{D}}$. Considering a sequence of N independent and identically distributed (i.i.d.) random variables, we formally define a fuzzing campaign $\mathcal{F}$ as: $\mathcal{F} = \{X_n \mid X_n \in \boldsymbol{\mathcal{D}}\}_{n=1}^{N}$. The input space $\boldsymbol{\mathcal{D}}$ is partitioned into S (potentially overlapping) subdomains $\{\mathcal{D}_i\}_{i=1}^{S}$, each representing a distinct *coverage element*. An input $X_n \in \mathcal{F}$ is said to discover a *new* coverage element $\mathcal{D}_i$ if $X_n \in \mathcal{D}_i$ and no previous input X_m for $m < n$ has been drawn from $\mathcal{D}_i$, i.e., $\mathcal{D}_i$ is encountered for the first time.

B. Bernoulli Product Model

In fuzzing, each input may belong to one or more coverage elements. The STADS framework represents this as *sampling-unit-based incidence data* [14], [15], where a sampling unit aggregates all inputs generated within a fixed time interval. This aggregation reduces the overhead of tracking fine-grained coverage for every individual input. The underlying probabilistic model is the *Bernoulli product model*.

The incidence data records whether a sampling unit has covered a given coverage element. Let π_i denote the probability that a sampling unit covers element $\mathcal{D}_i$, assumed constant across all randomly selected sampling units. Note that, in general, the sum of all π_i does *not* equal one.

During a fuzzing campaign, let t sampling units be recorded. The incidence data can be represented as an element-by-sampling-unit matrix $W = \{W_{ij} \mid i = 1, \ldots, S; j = 1, \ldots, t\}$ with S rows and t columns, where $W_{ij} = 1$ if element i is covered in sampling unit j, and $W_{ij} = 0$ otherwise.

The incidence frequency Y_i denotes the number of sampling units covering element $\mathcal{D}_i$, i.e., $Y_i = \sum_{j=1}^{t} W_{ij}$. Elements that has not been covered by any sampling unit will have $Y_i = 0$, i.e. zero incidence frequency.

Assuming the detection probabilities $(\pi_1, \pi_2, \ldots, \pi_S)$, each entry W_{ij} of the incidence matrix is modeled as an independent Bernoulli random variable with success probability π_i. The joint probability distribution of the incidence matrix is:

$$\begin{aligned} P(W_{ij} = w_{ij}; i = 1, 2, \ldots, S, j = 1, 2, \ldots, t) \\ = \prod_{j=1}^{t} \prod_{i=1}^{S} \pi_i^{w_{ij}} (1 - \pi_i)^{1 - w_{ij}} \\ = \prod_{i=1}^{S} \pi_i^{y_i} (1 - \pi_i)^{t - y_i}. \end{aligned} \tag{1}$$

The marginal distribution of the incidence frequency Y_i for the i-th coverage element follows a binomial distribution with t trials and success probability π_i:

$$P(Y_i = y_i) = \binom{t}{y_i} \pi_i^{y_i} (1 - \pi_i)^{t - y_i}, \qquad i = 1, 2, \ldots, S. \tag{2}$$

[1] Blackbox random testing has been the workhorse for reliability studies historically [10].

[2] STADS—Software Testing As Discovery of Species

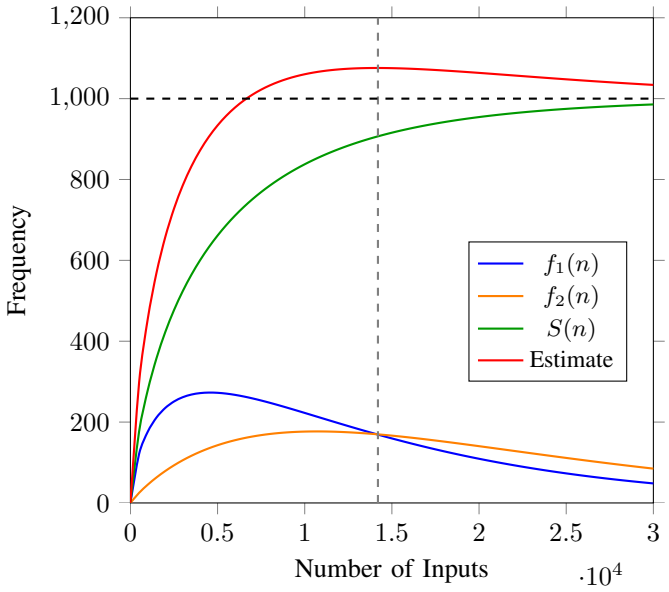

Fig. 1: Visualization of $f_1 f_2$ equilibrium

Let the incidence frequency counts be $(f_0, f_1, \ldots, f_t)$, where f_k denotes the number of elements covered in exactly k sampling units, for $k = 0, 1, \ldots, t$. In particular, f_1 counts the *singleton* elements (exercised by only one sampling unit), and f_2 counts the *doubleton* elements (exercised by two sampling units). The unobserved zero frequency f_0 represents elements not covered by any of the t sampling units. Consequently, the total number of covered elements in the campaign is $S(t) = \sum_{i>0} f_i$, and $S(t) + f_0 = S$.

C. Adaptive Bias in Greybox Fuzzing

In blackbox fuzzing, the detection probabilities π_i for each species remain invariant, as the fuzzer does not use coverage feedback. In contrast, greybox fuzzing employs lightweight instrumentation to guide input generation, resulting in detection probabilities that evolve as the campaign progresses [6]. Empirical studies indicate that the probability of discovering a given species in a greybox fuzzer $\mathcal{F}_G$ increases over time, thereby enhancing overall effectiveness relative to an equivalent blackbox fuzzer $\mathcal{F}_B$ [8].

This adaptive behavior introduces *adaptive bias*, which directly impacts the observed incidence frequencies f_k as described in Equation 2. Formally, adaptive bias can be quantified by the differential rate of species discovery between $\mathcal{F}_G$ and $\mathcal{F}_B$ across successive inputs [8]. Let $S_G(n)$ and $S_B(n)$ denote the cumulative number of species discovered by $\mathcal{F}_G$ and $\mathcal{F}_B$ after the n^{th} input, respectively. The adaptive bias Γ at step n is then defined as:

$$\Gamma(n) = [S_G(n+1) - S_G(n)] - [S_B(n+1) - S_B(n)] \quad (3)$$

Böhme et al. [8] shows that adaptive bias diminishes over time as the greybox campaign progresses.

D. Estimating f_0 at $f_1 f_2$ Equilibrium

Recent research [9] proposes a criterion for obtaining an upper bound on the number of undiscovered species, f_0, in a fuzzing campaign. Because this condition is derived from the observed singleton and doubleton counts (f_1 and f_2), we refer to it as the *singleton–doubleton equilibrium* or the $f_1 f_2$ equilibrium. The formal definition is given below.

Let us define $f_i(n)$ as the frequency of species observations in n observations such that $f_0(n)$ is the number of unobserved species, $f_1(n)$ as the number of singleton species etc.

> *For an arbitrary species distribution* $\{\pi_i\}_{i=1}^{S}$, *where* $i : 1 \leq i \leq S$, *there exists* n_1 *such that* $f_1(n_1) = f_2(n_1)$. *Then, for all* $n \geq n_1$, *it holds that* $f_1(n) \geq f_0(n)$.

That is, at n_1 observations we have the $f_1 f_2$ equilibrium. Theorem 1 from prior work [9] given below, is the special case in which species follow a uniform distribution.

Theorem 1. *Assume that the probability of a generated input belonging to species* i, $1 \leq i \leq S$, *is* uniform, *i.e.,* $\pi_i = \frac{1}{S}$. *If we select* n_1 *such that* $f_1(n_1) = f_2(n_1)$, *then for all* $n \geq n_1$, *it holds that* $f_1(n) \geq f_0(n)$.

The researchers hypothesize that although species distributions may vary from purely uniform distribution, *empirically*, coverage elements are distributed such that the criterion applies in most practical scenarios. While it may be possible to construct specific instances where $f_1 f_2$ equilibrium cannot occur, the STADS [6] framework implicitly assumes that such patterns are rare, and does not significantly impact the results.

In fuzzing campaigns, the $f_1 f_2$ equilibrium manifests only after prolonged execution. Prior work [9] hypothesizes that, at this point, the impact of adaptive bias is minimal. The key focus of this report is to validate this hypothesis.

Figure 1 demonstrates the $f_1 f_2$ equilibrium visually. In the figure, The total number of species is 1000. The $S(n)$ is the total number of species found so far. At the $f_1 f_2$ equilibrium f_1, that is the number of species discovered only once becomes that upper-bound for f_0, that is the number of species yet to be discovered. We also include an *Estimate* from the Jacknife estimator, which demonstrates the inflection at $f_1 f_2$ equilibrium. That is, the upperbound of the species estimate produced at $f_1 f_2$ equilibrium remains an upperbound for the remainder of the campaign.

E. Generating Fuzzing Subjects With Known Reachability

We use synthetic parser generation [11] for generating large fuzzing benchmarks with rich control-flow structure.

Formally, any program written in the structured programming style can be represented as a static control-flow graph, with extra annotations representing the data-flow and constraints. Such a control-flow graph represents all paths that can be traversed in the program, and along with the annotations represent an equivalent executable representation of the program. That is, control-flow graphs represent program skeletons. Hence, if we can produce arbitrary control-flow

graphs, we can produce an unbiased population of the program skeletons representing all possible path diversity.

From Control-Flow to Grammar. A control-flow graph is made up of the following sub-structures.

- **Statement sequences**, representing sequential execution;
- **Conditionals** for branching (e.g., `if-then-else`);
- **Loops** representing iteration (e.g., `while` loops);
- **Subroutine calls** for invoking other procedures.

These control-flow structures map to the following structures in LL(1) grammar:

- Conditionals such as `if C: A else: B` map to $\langle \bullet \rangle \rightarrow \langle C \rangle\ \langle IF^i \rangle$ $\langle IF^i \rangle \rightarrow$ `'T'` $\langle A \rangle \mid$ `'F'` $\langle B \rangle$;
- Loops such as `while C: B` map to $\langle \bullet \rangle \rightarrow \langle C \rangle\ \langle L^i \rangle$ $\langle L^i \rangle \rightarrow$ `'T'` $\langle B \rangle\ \langle C \rangle\ \langle L^i \rangle \mid$ `'F'`;
- Procedure definitions translate to nonterminals.

That is, LL(1) grammars are isomorphic to structured program control-flows. Next, we show how to leverage this property.

There is a well-known approach [16] generating recursive-descent parsers from LL(1) grammars. Handwritten parsers are subjects in many fuzzing contexts. Hence, we generate arbitrary LL(1) grammars, convert them to recursive-descent parsers, and use these generated programs for evaluation.

Context-Free Grammars. A context-free grammar (aka grammar) defines a language using *terminal* and *nonterminal* symbols. Consider the following example:

$$\langle E \rangle \rightarrow \langle D \rangle\ \langle Es \rangle$$

$$\langle Es \rangle \rightarrow \texttt{'+'}\ \langle D \rangle\ \langle Es \rangle \mid \epsilon$$

$$\langle D \rangle \rightarrow \texttt{'0'} \mid \texttt{'1'} \mid \ldots$$

The *terminal symbols* are the symbols of the language described by the grammar. For example, in the above grammar, `'0'` is a terminal symbol. The grammar also contains *nonterminal symbols* which are named constructs that are expanded into other terminal or other nonterminal symbols. For example, $\langle E \rangle$ is a nonterminal symbol.

Nonterminal symbols are expanded into other symbols according to a set of *production rules* (also called *rules*), which together *define* the nonterminal. The definition of the nonterminal $\langle Es \rangle$ in the above example is given by $\langle Es \rangle \rightarrow$ `'+'` $\langle D \rangle\ \langle Es \rangle \mid \epsilon$ which consists of two rules for expanding $\langle Es \rangle$. LL(1) grammars have three key properties:

1) No left recursion (direct or indirect), which ensures the parser can predict the next production from a single token.
2) Left-factored, that is, no two productions for the same nonterminal should start with the same terminal.
3) Single lookahead is sufficient to choose between two production rules.

Transforming Grammar to Executable Programs. Given any such LL(1) grammar, we can automatically construct a recursive-descent parser by applying the following rules:

- Translate each nonterminal into a procedure (e.g., $\langle D \rangle$ becomes `parse_D`).
- Implement each rule inside the corresponding procedure.
- Use the lookahead token to choose which rule to apply.
- Replace simple left-to-right recursion with a `while` loop when appropriate.

By following this process, we obtain a parser that can parse any sentence in the grammar. For example, given the grammar for for `'E'` we discussed previously, we can translate this to:

Algorithm 1 Expression Parser

```
1:  function parseE
2:      parseD
3:      while lookahead = '+' do
4:          consume('+')
5:          parseD
6:      return node

7:  function parseD
8:      token ← lookahead
9:      if token ≠ null and isdigit(token) then
10:         consume(token)
11:     else
12:         raise Error
```

Controlling the Program Complexity. While generating grammars, we have several mechanisms to precisely control the complexity. These include:

- The number of nonterminals (corresponding to the number of procedures),
- The number and length of production rules (corresponding to branching complexity),
- The depth and type of recursion (direct, indirect, or linear recursion),
- The inclusion of unreachable nonterminals or dead code to simulate partially unreachable programs.

These parameters allow us to generate large structured parser programs with complex control-flows.

III. Execution Plan

This section outlines the research methodology.

A. Research Questions

The primary objective of this study is to assess the influence of adaptive bias on state-of-the-art coverage saturation estimators. Prior work suggests that systematic underestimation may arise in greybox fuzzing [7], but it remains unclear to what extent this effect is attributable to adaptive heuristics rather than inherent statistical limitations. This motivates two experimental checks: (1) whether the adaptive bias can be empirically quantified using parallel greybox and blackbox fuzzing campaigns initialized from identical seeds as explained in Section II-C, and (2) whether adaptive bias demonstrably diminishes over time, as theorized by Böhme et al. [8].

Prior work [9] hypothesizes that once the campaign reaches the $f_1 f_2$ equilibrium, the influence of adaptive bias becomes

negligible because the utility of coverage feedback is insignificant. This implies that at the $f_1 f_2$ equilibrium, the maximum reachable coverage estimates obtained from a greybox fuzzing campaign should be accurate. Since we lack a reliable ground truth for any real world program, this hypothesis can't be tested directly. Hence, we propose an alternative.

We note that it is possible to turnoff the coverage feedback in typical greyboxfuzzers. If coverage feedback was the cause of adaptive bias, then, fuzzing campaigns with no coverage feedback should produce a reliable estimate of the maximum reachable coverage albeit with a larger variance due to the limited coverage obtainable in a given amount of time. If (a) the maximum reachable coverage estimators are reliable when adaptive bias is minimal and (b) adaptive bias is minimal at $f_1 f_2$ equilibrium, then at that point, the maximum reachable coverage estimate from greybox and blackbox campaigns must overlap. We propose to test this hypothesis.

RQ1: Does the maximum reachable coverage estimates from blackbox and greybox fuzzing campaigns converge at $f_1 f_2$ equilibrium?

If the maximum reachable coverage estimates converge, then it is evidence that (a) coverage feedback is responsible for the adaptive bias, and (b) its influence on reachability estimate can be minimized by waiting until the $f_1 f_2$ equilibrium. Non-converge suggests either that (a) the maximum reachable coverage estimates are unreliable or (b) the $f_1 f_2$ equilibrium does not indicate that the coverage feedback is negligible.

If the maximum reachable coverage estimates do not coverage at $f_1 f_2$ equilibrium, we can determine the validity of (b) by simply continuing the campaign to verify if the maximum reachable coverage estimates are tending to converge at a later time. However, it is possible that the reliability of maximum reachable coverage estimators is impacted by an unknown factor resulting in an estimate that is unreliable. Even if the estimates do converge, it is possible that the predicted maximum reachable coverage estimates is incorrect.

Hence, to assess estimator accuracy, we require subjects with a reliable ground truth for saturation. Recent work [11] suggests that *recursive descent parsers* generated from arbitrary context-free LL(1) grammars are good subjects for evaluating maximum reachable coverage estimators. All program elements in such parsers can be reached by execution, and hence provide a labeled ground truth for maximum reachable coverage estimation. We therefore can leverage arbitrarily generated recursive descent parsers as synthetic subjects with known saturation levels. If maximum reachable coverage estimators are accurate, such estimates should have significant overlap with the ground truth in such generated parsers at the equilibrium point. We propose to test this hypothesis with the second research question. We note that it is not clear if the adaptive bias monotonically reduces over time. Indeed, adaptive bias may shrink, plateau, or increase again during a campaign. If this happens, this should be observable in the convergence behavior of estimates from blackbox and greybox fuzzers.

RQ2: Are maximum reachable coverage estimators accurate at the $f_1 f_2$ equilibrium when evaluated against ground truth?

These are the potential outcomes from this experiment:

1) The estimators predict the maximum reachable coverage of recursive-descent programs accurately, and the maximum reachable coverage estimates from blackbox and greybox fuzzing campaigns coincide at the $f_1 f_2$ equilibrium.
2) The estimators predict the maximum reachable coverage of recursive-descent programs accurately, but the maximum reachable coverage estimates from blackbox and greybox fuzzing campaigns do not coincide at the $f_1 f_2$ equilibrium.
3) The estimators do not predict the maximum reachable coverage of recursive-descent programs accurately, and the estimates from blackbox and greybox fuzzing campaigns do not coincide at the $f_1 f_2$ equilibrium, but asymptotically (i.e, there is evidence that they will converge at a later point).
4) The estimators do not predict the maximum reachable coverage of recursive-descent programs accurately, and the maximum reachable coverage estimates from blackbox and greybox fuzzing campaigns do not coincide even asymptotically.
5) The estimators do not predict the maximum reachable coverage of recursive-descent programs accurately even asymptotically, but the estimates from blackbox and greybox fuzzing campaigns coincide at the $f_1 f_2$ equilibrium.

If (1) is observed, then it is strong evidence that we can rely on coverage estimators at the equilibrium point, and that the minimal adaptive-bias point coincides with the equilibrium point. If (2) is observed, it is evidence that a second factor beyond adaptive-bias is at play. If (3) is observed, then it is evidence that minimal-bias is not at the equilibrium point. If (4) is observed, the it is possible that the reachability estimator is unable to provide reliable estimates. We do not expect (5), to occur, but if it occurs, we will need to reevaluate our assumptions.

B. Coverage Saturation Estimators

We employ twelve state-of-the-art biostatistical species–richness estimators that constitute the current foundation of maximum reachable coverage estimation in fuzzing [7], [11], [17]. These estimators originate from ecological statistics and have been systematically adapted to model coverage saturation in software testing. Owing to their mathematical structure, many of them naturally cluster into families such as incidence-based, coverage-based, and extrapolation-driven estimators. Table I summarizes the full set of estimators considered in this study together with the notation used throughout the paper.

For complete mathematical definitions and derivations, refer to the prior work [7], [17].

TABLE I: Coverage Saturation Estimators (source [11])

Estimator class	Estimator	Notation
Chao-type	Chao2 [18]	$\hat{S}_{\text{Chao2}}$
	Bias corrected Chao2 (Chao2_bc) [19]	$\hat{S}_{\text{Chao2\_bc}}$
	Improved Chao2 (iChao2) [20]	$\hat{S}_{\text{iChao2}}$
Jackknife	First-order Jackknife (JK1) [21]	$\hat{S}_{\text{JK1}}$
	Second-order Jackknife (JK2) [21]	$\hat{S}_{\text{JK2}}$
Incidence-based coverage estimators (ICE)	ICE [22]	$\hat{S}_{\text{ICE}}$
	ICE1 [23]	$\hat{S}_{\text{ICE1}}$
Zelterman estimator	Zelterman [24]	$\hat{S}_{\text{Zelterman}}$
Bootstrap estimator	Bootstrap [25]	$\hat{S}_{\text{Bootstrap}}$
Frequency-based	Chao-Bunge [26]	$\hat{S}_{\text{Chao-Bunge}}$
Unconditional nonparametric maximum likelihood estimator	UNPMLE [27]	$\hat{S}_{\text{UNPMLE}}$
Penalized nonparametric maximum likelihood estimator	PNPMLE [28]	$\hat{S}_{\text{PNPMLE}}$

C. Fuzzer, Subject Programs, and Campaigns

Our experimental study consists of executing parallel fuzzing campaigns on a diverse set of subject programs over extended time horizons. Below, we describe the core components of our setup.

Fuzzer: We employ AFL++ [29], the current state-of-the-art greybox fuzzer widely used in empirical research. AFL++ provides a well-defined mutation and scheduling mechanism that closely aligns with the STADS model in Section II, making it suitable for controlled statistical analysis.

Since our study requires running parallel greybox and blackbox fuzzing campaigns under identical initial conditions, we use AFL++ in both modes while leveraging its `dumb mode` functionality. Specifically, enabling the `-n` flag disables coverage-guided mutations, thereby yielding a true blackbox baseline that is fully comparable to the corresponding greybox campaign. While greybox fuzzing is significantly more efficient than blackbox fuzzing in discovering new coverage, both modes target the same population of reachable coverage elements in the asymptote. Differences between them therefore reflect discovery dynamics rather than fundamentally different coverage populations, and hence, should impact the width of confidence intervals rather than the point estimates.

Programs: To evaluate estimator accuracy against known ground-truth reachability, we employ program generation [11] to produce a diverse corpus of recursive descent parsers in C with known reachability. Recursive-descent parsers are used as controlled experimental artifacts in which reachability is known by construction. While the semantic constraints, and data-flow of real programs are not captured, this construction represents the control-flow of any possible program written in the structured programming style. This allows us to evaluate estimator accuracy against known maximum reachable coverage, which is not possible for real-world programs. For this study, we will generate ten parsers spanning a wide range of sizes, chosen to reflect the scale of real-world programs. Each parser will be constructed to exhibit complex control-flow behavior—including loops, conditionals, direct recursion, indirect recursion, and linear recursion—to ensure realistic fuzzing difficulty. All generated benchmarks will be validated for compatibility with AFL++ and for suitability in long-running fuzzing experiments.

While similarity in size to real-world programs is necessary, it is insufficient for establishing comparability. The structural composition of control-flow features—and, equivalently, the induced species distribution—must also reflect the statistical characteristics observed in real software. To evaluate this, we will compare our generated parsers against real-world programs using species counts, empirical species-frequency distributions, and distributional density measures such as Shannon entropy. Using publicly available data from [7], Table II reports Poisson and Exponential goodness-of-fit results for species distributions obtained after 7-day AFL++ fuzzing campaigns, showing a consistent preference for the Exponential distribution across subjects. By incorporating our parser-generated subjects into this analysis, we will assess whether their species distributions exhibit similar statistical behavior, thereby validating their suitability as synthetic yet realistic benchmarks.

Campaign Length and Trials: Each target will first be fuzzed for seven wall-clock days, following the longest standard evaluation window in FuzzBench and recent reachability studies such as [7]. For statistical robustness, we conduct $K = 30$ independent trials per target. However, prior work [7] has demonstrated that almost all the real-world programs exhibit no clear coverage saturation even after seven days of fuzzing. Should a campaign fail to reach at least the $f_1 f_2$ equilibrium point within this window, we will extend the fuzzing duration accordingly to at most double the original time.

Initial Seed Corpus: Since the generated benchmark programs do not have an established seed corpora for fuzzing, we initialize each campaign with syntactically valid inputs derived from the grammars used to generate the parsers. A fresh seed corpus is produced for each trial to maintain independence across fuzzing runs.

IV. Evaluation Metrics

We employ distinct evaluation metrics for RQ1 and RQ2, reflecting their different analytical objectives. Let $\hat{S}$ denote an estimator of maximum reachable coverage and S the true (ground-truth) maximum reachable coverage.

A. Comparing Maximum reachable coverage (RQ1).

For evaluating RQ1, we will obtain $\hat{S_G}$ from greybox campaign, and compare it to $\hat{S_B}$ from the blackbox campaign.

Mean Bias for greybox-blackbox Comparison. At campaign lengths n_g and n_b for the greybox and blackbox fuzzers respectively, we define the estimator bias as the difference between their point estimates $\hat{S}$. To account for stochastic variation across runs, we compute the average *relative* bias over K independent trials:

$$\textit{mean bias}(n_g, n_b) = \frac{1}{KS} \sum_{i=1}^{K} \left(\hat{S}_{iG}(n_g) - \hat{S}_{iB}(n_b) \right). \quad (4)$$

TABLE II: Poisson and Exponential model fits for species distributions

Subject	Poisson					Exponential				
	λ	λ_{se}	RMSE	AIC	BIC	rate	$rate_{se}$	RMSE	AIC	BIC
ffmpeg	2451047	9.954911	7.046377e+14	Inf	Inf	4.079890e-07	2.594239e-09	0.4755227	777213.06	777221.18
freetype2	65098403	90.507290	7.021900e+12	Inf	Inf	1.536136e-08	1.723170e-10	0.3961756	301851.48	301858.46
gif2png	33797182	254.695564	5.730000e+10	Inf	Inf	2.958827e-08	1.296285e-09	0.2276121	19108.00	19112.25
jasper	39893368	69.546620	8.247000e+11	Inf	Inf	2.506682e-08	2.760101e-10	0.1714684	305206.38	305213.40
jsoncpp	109285859	309.213802	2.613000e+11	Inf	Inf	9.150315e-09	2.706530e-10	0.2624239	44600.67	44605.71
readelf	46164072	107.970338	6.591000e+12	Inf	Inf	2.166187e-08	3.442297e-10	0.6310915	147691.88	147698.17

Note that the comparison may be made between estimates at arbitrary points in the greybox and blackbox campaigns; in particular, it is not required that $n_g = n_b$.

B. Measuring Maximum reachable coverage (RQ2).

For evaluating RQ2, we will obtain $\hat{S}$ from generated recursive descent programs, and compare obtained $\hat{S}$ to S.

Mean Bias for Ground-Truth Comparison. At campaign (either greybox or blackbox) length n, the bias of an estimator is defined as the difference between its point estimate $\hat{S}$ and the known ground truth S. To account for stochastic variation across runs, we compute the average *relative* bias over K independent trials:

$$\textit{mean bias}(n) = \frac{1}{KS}\sum_{i=1}^{K}\left(\hat{S}_i(n) - S\right). \tag{5}$$

Smaller absolute values indicate higher estimation accuracy.

Variance Across Trials. Variance quantifies the stability of an estimator across repeated campaigns. For each estimator, we report the variance (or *imprecision*[3]) of $\hat{S}$ over K trials. Lower variance indicates greater reliability and robustness.

Confidence-Interval (CI) Coverage. For estimators that provide confidence intervals, we assess *CI coverage*, defined as the proportion of trials in which the true saturation level S falls within the reported interval. Higher coverage indicates well-calibrated uncertainty quantification.

C. Other Observable Metrics

Measuring Adaptive Bias. We can quantify the *adaptive bias* $\Gamma_i(n)$ in the ith trial at input count n using the definition in Equation 3. This metric reflects the differential rate of species discovery between parallel greybox and blackbox campaigns at matching campaign progress. Given K independent trials, we compute the mean adaptive bias as,

$$\bar{\Gamma}(n) = \frac{1}{K}\sum_{i=1}^{K}\Gamma_i(n). \tag{6}$$

To ensure that $\Gamma(n)$ reflects only differences in detection probabilities and not artifacts of differing throughput, we will verify that the input-generation rates of the greybox and blackbox fuzzers remain closely aligned throughout the campaign.

To evaluate the performance of statistical estimators of maximum reachable coverage we employ the following metrics.

[3]Imprecision [7] is defined as the variance of individual bias values.

V. Related Work

Determining reachability is a foundational problem in software testing [7]. Establishing maximum reachability level is particularly challenging for large, complex code bases [30], where control-flow interactions, deep nesting, and diverse input formats hinder precise analysis. Static analysis techniques, including abstract interpretation and symbolic execution have been investigated to approximate reachable coverage, but these often suffer from systematic over- or under-approximation [7], [31]. For instance, Nikolić and Spoto [32] introduced an abstract domain for reasoning about reachability of program variables, yielding conservative over-approximations suitable primarily for identifying unreachable code. Similarly, Mikoláš et al. [33] proposed annotating code with explicit reachability conditions to detect unexecutable regions.

To assess reachability in dynamic analysis, both constraint-solving techniques such as SMT and data-driven estimation methods have been explored. Naus et al. [34] proposed a technique for automatically generating preconditions to trigger specific post-conditions (e.g., bugs) using low-level code analysis. Similarly, Liew et al. [35] demonstrated how to encode SMT formulas within coverage-guided fuzzers to discover inputs that reach targeted program locations. In contrast, statistical approaches directly attempt to estimate the test effectiveness (aka maximum reachability) through observed behaviors. The pioneering STADS framework [6] introduced a suite of bio-statistical estimators by modeling fuzzing as a statistical sampling process. These estimators have since been evaluated for their reliability in mutation analysis and fuzzing [11], [17]. However, applying them to greyboxfuzzing introduces *adaptive bias*, as feedback-driven input selection violates the assumption of an invariant underlying species distribution [6], [36]. While prior work has examined adaptive bias for measures such as discovery probability and coverage rate [8], [37], [38], its effect on *coverage-saturation estimators* has received comparatively less attention. Recent evidence indicates that adaptive bias can substantially distort these estimators, highlighting the need for *structure-aware* estimators that incorporate program-level structural information rather than relying solely on structure-agnostic sampling [39].

Synthetic benchmarks via program synthesis has been attempted before. Fuzzle [40] is a fuzzer evaluation benchmark constructed by encoding maze moves sequences as function calls chains. However, mazes do not closely resemble real-

world programs: their structural complexity is limited by the constrained set of possible moves, and solving a maze typically reduces to discovering a single valid path. In contrast, recursive-descent parsers are real-world programs and constitute a major class of targets in practical fuzzing. Olympia [41], built on Fuzzle, generates Solidity contracts from synthetic mazes, but inherits similar structural limitations. More recently, researchers [11] proposed generating recursive-descent parsers derived from LL(1) grammars as synthetic yet realistic substitutes for real-world fuzzing benchmarks such as `freetype2` in FuzzBench [42]. These parser-based benchmarks offer richer control-flow complexity and better reflect the structural characteristics of real-world software.

VI. Threats to Validity

As with any empirical investigation, our analysis of adaptive bias and coverage-saturation estimation in greybox fuzzing is subject to several threats to validity.

Internal validity concerns systematic errors that may influence our observations. Implementation or instrumentation defects, as well as unintended biases introduced during data processing, may affect our results. To mitigate these risks, we rigorously validate our tooling and apply consistency checks throughout data collection and preparation. We also perform multiple independent trials to reduce variability stemming from seed selection, fuzzer heuristics, and grammar-generation parameters. Machine load can influence execution speed and thus the number of generated inputs; this becomes especially critical in our parallel greybox and blackbox campaigns, where comparable execution rates are essential. We do not claim that results from recursive-descent parsers alone generalize to all software. Instead, the study combines real-world benchmarks without ground truth and synthetic benchmarks with known reachability, allowing us to reason about estimator behavior across both settings. We minimize these threats by executing experiments on controlled, isolated systems with monitored resource utilization, and by ensuring that parallel campaigns use machines with identical specifications under equivalent operating conditions.

External validity pertains to the generalizability of our findings. Our use of synthetic recursive-descent parsers may limit direct extrapolation to arbitrary real-world software. However, parser programs constitute a major class of real-world fuzzing targets and feature complex control-flow characteristics representative of broader software systems. Furthermore, AFL++ primarily targets C/C++ programs; thus, our results may not fully extend to languages with substantially different execution or memory models. Nonetheless, the control-flow structures captured in our synthetic benchmarks—loops, branching, and various forms of recursion—are widely present across programming languages.

Construct validity concerns whether our measurements faithfully capture the theoretical constructs under study. For RQ1, we quantify adaptive bias following the definition of [8], which models bias as the difference in expected species-discovery rates between parallel greybox and blackbox campaigns. This assumes that these differences arise primarily from feedback-driven adaptation rather than unrelated algorithmic effects. Although AFL++ incorporates several engineering optimizations, its core search strategy remains closely aligned with the STADS abstraction, suggesting that confounding influences on our adaptive-bias measurements are minimal. For RQ2, construct validity is inherently strong: estimator accuracy is evaluated directly against ground-truth reachability available for the synthetic benchmarks. Thus, the measured deviations reflect genuine estimator error rather than artifacts of the experimental setup.

VII. Future Work

This work investigates the impact of adaptive bias in greybox fuzzing on maximum reachable coverage estimators. Depending on our findings, one possible outcome is that the impact of adaptive bias diminishes sufficiently over long campaigns, yielding estimator accuracy that is acceptable for practical use in real-world fuzzing settings. However, to enable reliable estimation *throughout* a greybox campaign, particularly during early and mid-campaign phases where adaptive bias is most pronounced, future work should focus on developing principled bias-correction mechanisms. Similar correction strategies have been studied extensively in bio-statistics for related estimators [43], [44]. Adapting such techniques to the fuzzing context represents a promising direction for enabling robust, structure-aware estimators that remain valid under adaptive sampling dynamics.

VIII. Conclusion

Fuzzing campaigns require accurate estimation of maximum reachable coverage to ensure that resources are not wasted. However, adaptive bias due to the use of coverage feedback in modern fuzzers prevents accurate statistical estimation of maximum reachable coverage. Recent work hypothesizes that adaptive bias is minimized at the $f_1 f_2$ equilibrium, when f_1, the number of species observed exactly once, equals f_2, the number observed exactly twice. Rigorous evaluation of this hypothesis has been hindered by the lack of ground truth.

This registered report proposes two complementary approaches to bridge this gap. First, we compare maximum reachable coverage estimates between coverage-guided and purely random fuzzers on real-world benchmarks; since random fuzzers lack coverage feedback, agreement between estimates at the $f_1 f_2$ equilibrium would indicate that adaptive bias has been adequately minimized. Second, we validate estimates using synthetic programs with known maximum reachable coverage, where complex control flows mimic real-world complexity while providing objective ground truth.

Together, these studies will determine whether maximum reachable coverage estimates are reliable when the $f_1 f_2$ equilibrium criterion is satisfied, validating or refuting its use as a stopping criterion for fuzzing campaigns.

References

[1] V. J. Manès, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo, "The art, science, and engineering of fuzzing: A survey," *IEEE TSE*, vol. 47, no. 11, pp. 2312–2331, 2019.

[2] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 276–291, 2013.

[3] J. R. Horgan, S. London, and M. R. Lyu, "Achieving software quality with testing coverage measures," *Computer*, vol. 27, no. 9, 1994.

[4] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based greybox fuzzing as markov chain," ser. CCS '16, 2016, p. 1032–1043.

[5] A. Chao, C.-H. Chiu *et al.*, "Species richness: estimation and comparison," *Wiley StatsRef: statistics reference online*, vol. 1, p. 26, 2016.

[6] M. Böhme, "STADS: Software testing as species discovery," *ACM TOSEM*, vol. 27, no. 2, pp. 7:1–7:52, Jun. 2018.

[7] D. Liyanage, M. Böhme, C. Tantithamthavorn, and S. Lipp, "Reachable coverage: Estimating saturation in fuzzing," in *ICSE*, ser. ICSE, 2023, p. 371–383.

[8] M. Böhme, D. Liyanage, and V. Wüstholz, "Estimating residual risk in greybox fuzzing," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 230–241. [Online]. Available: https://doi.org/10.1145/3468264.3468570

[9] D. Liyanage, "Quantitative decision-making for automated software testing," Ph.D. dissertation, Monash University, 2023.

[10] D. Hamlet, "When only random testing will do," in *Proceedings of the 1st international workshop on Random testing*, 2006, pp. 1–9.

[11] D. Liyanage, N. Attanayake, Z. Luo, and R. Gopinath, "Assessing reliability of statistical maximum coverage estimators in fuzzing," *arXiv preprint arXiv:2507.17093*, 2025.

[12] H. L. Nguyen and L. Grunske, "Bedivfuzz: Integrating behavioral diversity into generator-based fuzzing," in *ICSE*, ser. ICSE, 2022, pp. 1–13.

[13] M. Böhme, V. Manès, and S. K. Cha, "Boosting fuzzer efficiency: An information theoretic perspective," in *ESEC/FSE*, ser. ESEC/FSE, 2020.

[14] R. K. Colwell, A. Chao, N. J. Gotelli, S.-Y. Lin, C. X. Mao, R. L. Chazdon, and J. T. Longino, "Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages," *Journal of plant ecology*, vol. 5, no. 1, pp. 3–21, 2012.

[15] A. Chao and R. K. Colwell, "Thirty years of progeny from chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling," *SORT: statistics and operations research transactions*, vol. 41, no. 1, pp. 0003–54, 2017.

[16] V. R. Pratt, "Top down operator precedence," in *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1973, pp. 41–51.

[17] K. Kuznetsov, A. Gambi, S. Dhiddi, J. Hess, and R. Gopinath, "Empirical evaluation of frequency based statistical models for estimating killable mutants," in *ESEM*, ser. ESEM, 2024, p. 61–71.

[18] A. Chao, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, pp. 783–791, 1987.

[19] A. Chao, R. L. Chazdon, R. K. Colwell, and T.-J. Shen, "A new statistical approach for assessing similarity of species composition with incidence and abundance data," *Ecology letters*, vol. 8, no. 2, pp. 148–159, 2005.

[20] C.-H. Chiu, Y.-T. Wang, B. A. Walther, and A. Chao, "An improved nonparametric lower bound of species richness via a modified good–turing frequency formula," *Biometrics*, vol. 70, no. 3, pp. 671–682, 2014.

[21] K. P. Burnham and W. S. Overton, "Estimation of the size of a closed population when capture probabilities vary among animals," *Biometrika*, pp. 625–633, 1978.

[22] S.-M. Lee and A. Chao, "Estimating population size via sample coverage for closed capture-recapture models," *Biometrics*, pp. 88–97, 1994.

[23] N. J. Gotelli and A. Chao, "Measuring and estimating species richness, species diversity, and biotic similarity from sampling data," *Encyclopedia of biodiversity*, pp. 195–211, 2013.

[24] D. Böhning, "Some general comparative points on chao's and zelterman's estimators of the population size," *Scandinavian Journal of Statistics*, vol. 37, no. 2, pp. 221–236, 2010.

[25] E. P. Smith and G. van Belle, "Nonparametric estimation of species richness," *Biometrics*, pp. 119–129, 1984.

[26] A. Chao and J. Bunge, "Estimating the number of species in a stochastic abundance model," *Biometrics*, vol. 58, no. 3, pp. 531–539, 2002.

[27] J. L. Norris and K. H. Pollock, "Non-parametric mle for poisson species abundance models allowing for heterogeneity between species," *Environmental and Ecological Statistics*, vol. 5, pp. 391–402, 1998.

[28] J.-P. Z. Wang and B. G. Lindsay, "A penalized nonparametric maximum likelihood approach to species richness estimation," *Journal of the American Statistical Association*, vol. 100, no. 471, pp. 942–959, 2005.

[29] M. Vanhauser, "Aflplusplus," 2025. [Online]. Available: https://aflplus.plus/

[30] T. D. LaToza and B. A. Myers, "Developers ask reachability questions," in *ICSE*, ser. ICSE, 2010, p. 185–194.

[31] M. F. Aniche, G. A. Oliva, and M. A. Gerosa, "Why statically estimate code coverage is so hard? a report of lessons learned," in *2015 29th Brazilian Symposium on Software Engineering*, 2015, pp. 185–190.

[32] D. Nikolić and F. Spoto, "Reachability analysis of program variables," *ACM TPLS*, vol. 35, no. 4, Jan. 2014.

[33] M. Janota, R. Grigore, and M. Moskal, "Reachability analysis for annotated code," in *ESEC/FSE*, ser. SAVCBS '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 23–30.

[34] N. Naus, F. Verbeek, M. Schoolderman, and B. Ravindran, "Low-level reachability analysis based on formal logic," in *International Conference on Tests and Proofs*. Springer, 2023, pp. 21–39.

[35] D. Liew, C. Cadar, A. F. Donaldson, and J. R. Stinnett, "Just fuzz it: solving floating-point constraints using coverage-guided fuzzing," in *ESEC/FSE*, ser. ESEC/FSE 2019, 2019, p. 521–532.

[36] M. Böhme, "Assurances in software testing: A roadmap," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019, pp. 5–8.

[37] S. Lee and M. Böhme, "Dependency-aware residual risk analysis," 2025.

[38] D. Liyanage, S. Lee, C. Tantithamthavorn, and M. Böhme, "Extrapolating coverage rate in greybox fuzzing," in *ICSE*, ser. ICSE, 2024.

[39] S. Lee and M. Böhme, "Statistical reachability analysis," in *ESEC/FSE*, ser. ESEC/FSE. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3611643.3616268

[40] H. Lee, S. Kim, and S. K. Cha, "Fuzzle: Making a puzzle for fuzzers," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.

[41] J. Chadt, C. Hochrainer, V. Wüstholz, and M. Christakis, "Olympia: Fuzzer benchmarking for solidity," ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2362–2365. [Online]. Available: https://doi.org/10.1145/3691620.3695352

[42] J. Metzman, L. Szekeres, L. Simon, R. Sprabery, and A. Arya, "Fuzzbench: an open fuzzer benchmarking platform and service," in *ESEC/FSE*, 2021, pp. 1393–1403.

[43] S. K. Thompson, "Adaptive cluster sampling: designs with primary and secondary units," *Biometrics*, pp. 1103–1115, 1991.

[44] M. Salehi and G. A. Seber, "Two-stage adaptive cluster sampling," *Biometrics*, pp. 959–970, 1997.