浙江大学 ZHEJIANG UNIVERSITY
浙江師範大學 ZHEJIANG NORMAL UNIVERSITY
CEPREI 中国赛宝实验室（工业和信息化部电子第五研究所）

Boosting Parallel Fuzzing with Boundary-Targeted Task Allocation and Exploration

Hong Liang Yijia Guo Haotian Wu Yifan Xia Yi Xiang
Zonghui Wang Yandong Gao Wenzhi Chen Shouling Ji

浙江大学网络系统安全与隐私实验室
NETWORK SYSTEM SECURITY & PRIVACY LAB

Introduction: What is Fuzzing?

Fuzzing

A widely adopted technique for discovering software vulnerabilities by generating diverse inputs to trigger unexpected program behavior.

Challenge:

- Modern software is complex and frequently updated.
- Single-instance fuzzing is too slow.

Solution:

- **Parallel Fuzzing**: Run multiple instances simultaneously to find more bugs faster.

Motivation: The Bottleneck of Redundant Exploration

The Core Problem

Parallel fuzzers waste resources exploring the **same code paths**, and this overlap does **not decline** over time.

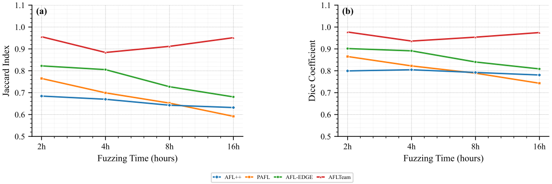


Figure: Task overlap on *tcpdump* remains high.

Motivation: The Bottleneck of Redundant Exploration

The Core Problem

Parallel fuzzers waste resources exploring the **same code paths**, and this overlap does **not decline** over time.

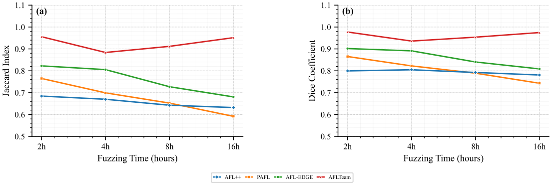


Figure: Task overlap on *tcpdump* remains high.

The Consequence

Lower overlap aligns with stronger coverage growth. **Duplicated exploration** is the primary bottleneck.

Table: Edge Coverage on *tcpdump*

Time	AFL++	PAFL	AFL-EDGE	AFLTeam
2h	**19,262**	15,956	15,971	15,089
4h	**21,516**	19,943	19,784	16,902
8h	24,075	**24,872**	23,788	20,777
16h	25,539	**26,021**	24,710	22,907

Motivation: Why Existing Strategies Fall Short

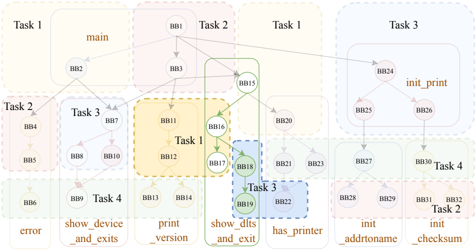
The Root Cause

There is a mismatch between where tasks are split and where overlap actually accumulates.

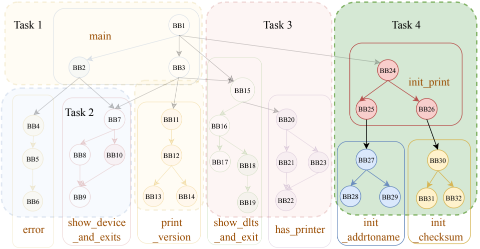
The Root Cause

There is a mismatch between where tasks are split and where overlap actually accumulates.

Mutually-Exclusive breaks control-flow locality by fragmenting contiguous code.



Structure-Aware ignores hard-to-reach areas while over-fuzzing common functions.

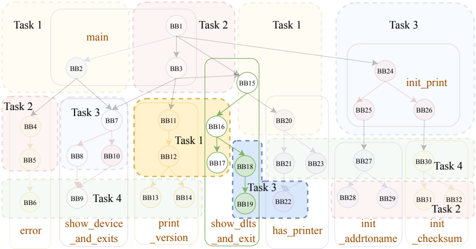


Motivation: Why Existing Strategies Fall Short

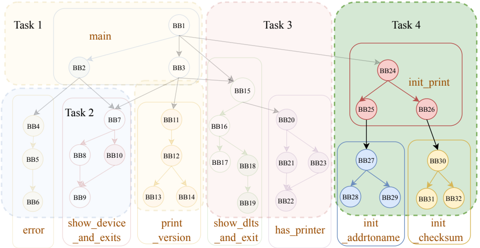
The Root Cause

There is a mismatch between where tasks are split and where overlap actually accumulates.

Mutually-Exclusive breaks control-flow locality by fragmenting contiguous code.



Structure-Aware ignores hard-to-reach areas while over-fuzzing common functions.



Instead of static, deep partitions, we should partition tasks at the **moving boundary** between covered and uncovered code.

FLEXFUZZ: A Boundary-Targeted Parallel Fuzzer

Our Solution: FLEXFUZZ

A novel parallel fuzzing system designed to minimize redundancy by focusing on the most promising areas for exploration.

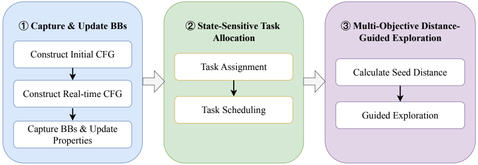


Figure: High-level pipeline of **FLEXFUZZ**.

Key Concepts:

- **Boundary Basic Blocks**: The frontier with the highest potential to reveal new paths.
- **Boundary-Sensitive Allocation**: Dynamic task assignment based on the evolving boundary.
- **Distance-Guided Exploration**: Focuses each instance on its assigned task area.

The FLEXFUZZ Framework

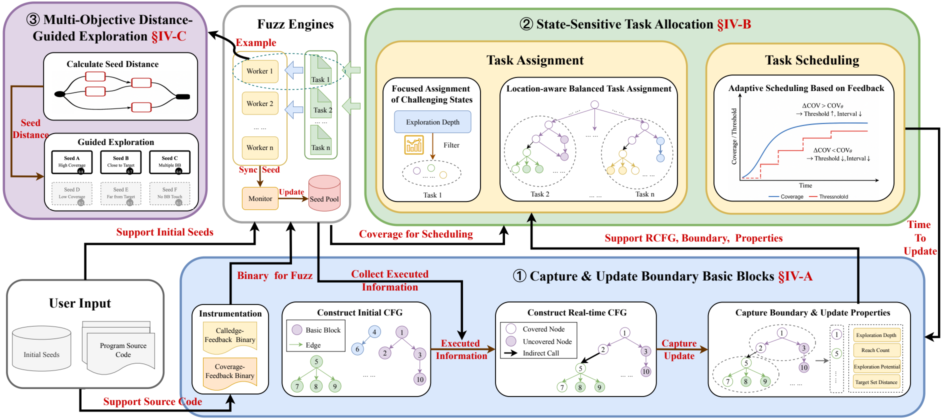


Figure: The core feedback loops of the **FLEXFUZZ** framework.

Module 1: Capture and Update Boundary

Defining the Boundary

A basic block is a **boundary basic block** if it is covered, but has at least one successor that remains uncovered.

Process:

- LLVM instrumentation builds an accurate RCFG, capturing indirect calls.
- The boundary is updated continuously as the fuzzer finds new seeds.

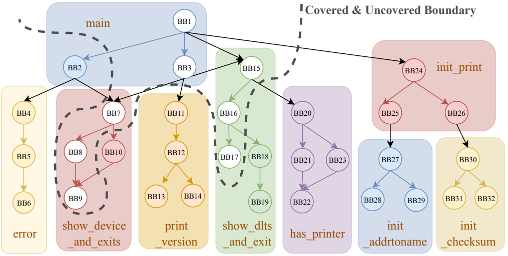


Figure: Real-time CFG. The gray dotted line marks the exploration boundary.

Module 2: Boundary-Sensitive Task Allocation

Challenge 1

How to divide tasks while preserving control-flow locality and adapting to the changing boundary?

Our Two-Part Solution:

1. **Focused & Balanced Task Assignment**
   - Challenging nodes get a dedicated task.
   - DFS-based traversal preserves locality and balances workload.
2. **Adaptive Task Scheduling**
   - Exponential backoff mechanism inspired by TCP congestion control.
   - Task update interval adjusts dynamically based on coverage growth.

Module 3: Multi-Target Distance-Guided Exploration

Challenge 2

How do we ensure each fuzzer instance remains focused on its assigned sub-task?

Our Approach:

- **Selective Targeting**: Prioritize **rare** boundary blocks (low reach counts) as targets.
- **Lightweight Distance Metric**:
  - Fine-grained, basic-block-level distance calculated on the RCFG.
  - Seed's distance = minimum distance from its execution path to any target.
- **Guided Exploration**:
  - Seeds closer to targets get higher selection probabilities and more mutation energy.

Evaluation Setup

Research Questions

RQ1: How does **FLEXFUZZ** perform vs. state-of-the-art parallel fuzzers?

RQ2: What are the contributions of **FLEXFUZZ**'s main components?

Baselines:

- AFL++ (parallel)
- K-Scheduler
- PAFL, AFL-EDGE
- AFLTeam, autofz

Benchmark & Environment:

- **9 large programs** (tcpdump, sqlite3, vim, ffmpeg, etc.)
- Each has over **40K branches**
- **10 cores**, 24h runs, 10 repeats

RQ1: Performance - Branch Coverage

Average covered branches after 24 hours:

Fuzzer	Branches	vs. AFL++
AFL++	24,482.60	–
PAFL	26,242.51	+2.18%
AFL-EDGE	24,560.90	+0.42%
AFLTeam	24,349.23	+0.20%
FLEXFUZZ	**32,842.00**	**+21.04%**

FLEXFUZZ achieves **21.04%** more coverage than AFL++, significantly outperforming all baselines.

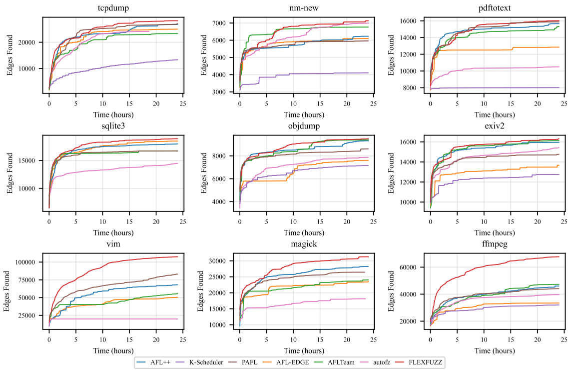


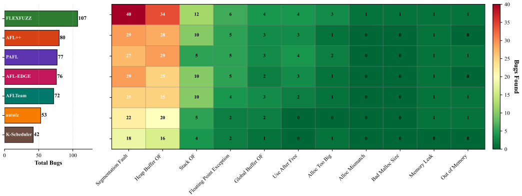
Figure: **FLEXFUZZ** consistently achieves higher coverage, especially on complex programs.

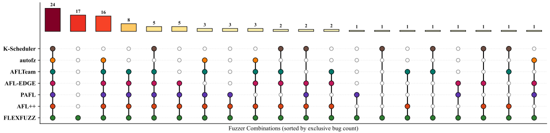
RQ1: Performance - Vulnerability Discovery

Finding More Bugs

> After manual deduplication, **FLEXFUZZ** finds significantly more unique vulnerabilities than all other fuzzers.

FLEXFUZZ finds **107 unique bugs** (33.75% more than AFL++) and discovers **17 bugs** missed by all other fuzzers.



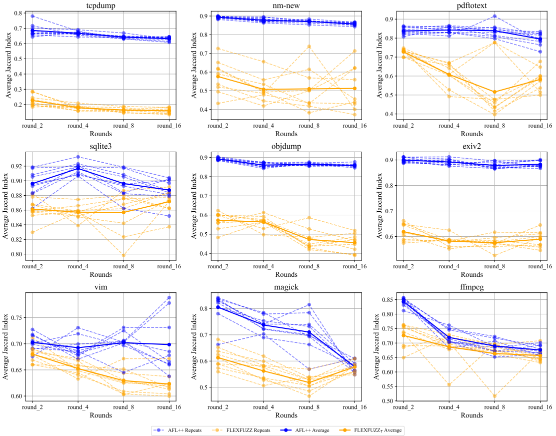


RQ2: Task Assignment Reduces Overlap

Metric: Average Jaccard Index

Measures the similarity of covered code between pairs of workers. A **lower index** means less overlap and more efficient, diverse exploration.

FLEXFUZZ$_T$ (task assignment only) consistently maintains a lower Jaccard Index than AFL++, confirming its effectiveness at reducing redundant work.

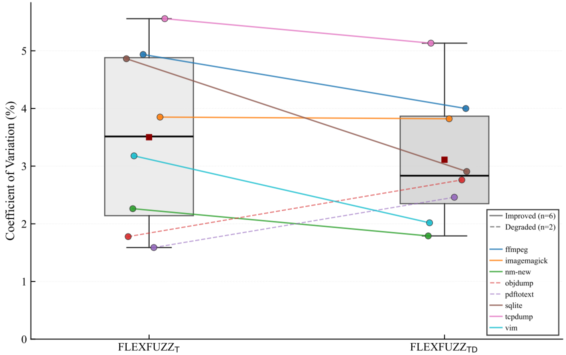


RQ2: Adaptive Scheduling Improves Stability

Metric: Coefficient of Variation (CV)

Measures the relative dispersion of coverage results across 10 independent trials. A **lower CV** indicates more stable and repeatable performance.

FLEXFUZZ$_{TD}$ (with adaptive scheduling) shows lower CV than **FLEXFUZZ**$_{T}$ (fixed scheduling), proving that dynamic updates lead to more stable outcomes.



RQ2: Guided Exploration Hits Its Targets

Evaluating Fuzzing Focus

We measured the percentage of newly generated seeds that successfully trigger the "rare" target basic blocks within their assigned task.

Key Result

The multi-target, distance-guided exploration is highly effective. It increased the proportion of new seeds hitting their intended rare targets from an average of **14.4% to 45.5%**.

This demonstrates that fuzzer instances are successfully directed toward the most challenging parts of their tasks.

Conclusion

Summary

FLEXFUZZ tackles redundant exploration by focusing workers on the moving boundary of code coverage.

Key Contributions:

- A novel **boundary-targeted framework** focusing on promising code regions.
- A **boundary-sensitive task allocation** scheme reducing overlap while preserving locality.
- A **multi-target distance-guided** strategy keeping workers focused on tasks.

Results

On average, **FLEXFUZZ** achieves: **21% higher coverage** than AFL++, and **1.34x more vulnerabilities**.

Thanks!

Hong Liang
hongliang@zju.edu.cn