Efficiently Rebuilding Coverage in Hardware-Assisted Greybox Fuzzing

Tai Yue[1,2,3], Yibo Jin[2], Fengwei Zhang*[2], Zhenyu Ning[4], Pengfei Wang*[3], Xu Zhou[3], and Kai Lu[3]

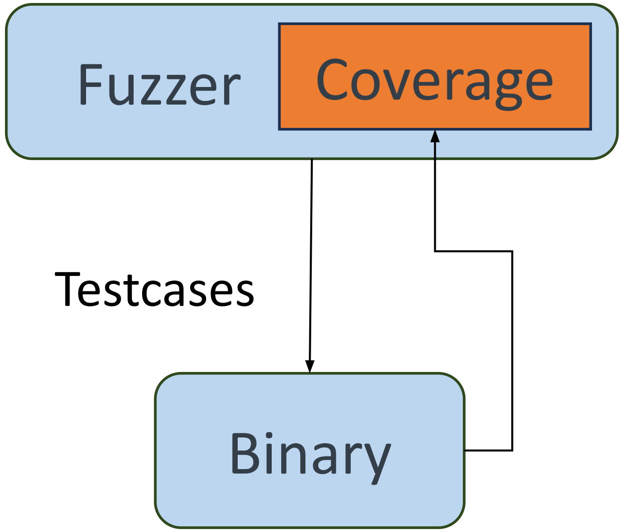
[1] Academy of Military Science,

[2] Southern University of Science and Technology, [3] National University of Defense Technology, [4] Hunan University

Rebuilding Coverage in Binary-only Fuzzing

Existing techniques in rebuilding coverage:

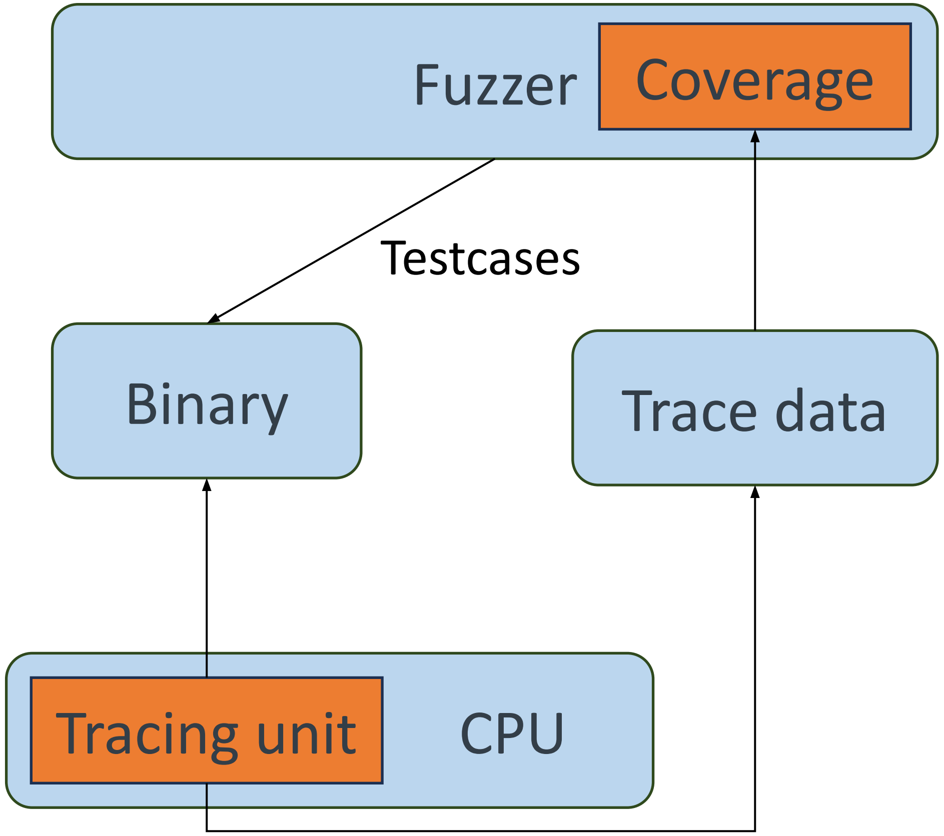
- Dynamic binary instrumentation (e.g., AFL-PIN, AFL-QEMU)
  - Heavy overhead
- Static binary rewriting (e.g., RetroWrite)
  - Additional prerequisites for binaries
- Hardware tracing (e.g., PTFuzz, PTrix)
  - Negligible runtime overhead
  - Powerful tracing ability



Hardware-assisted Greybox Fuzzing

Hardware-assisted greybox fuzzing (HGF):

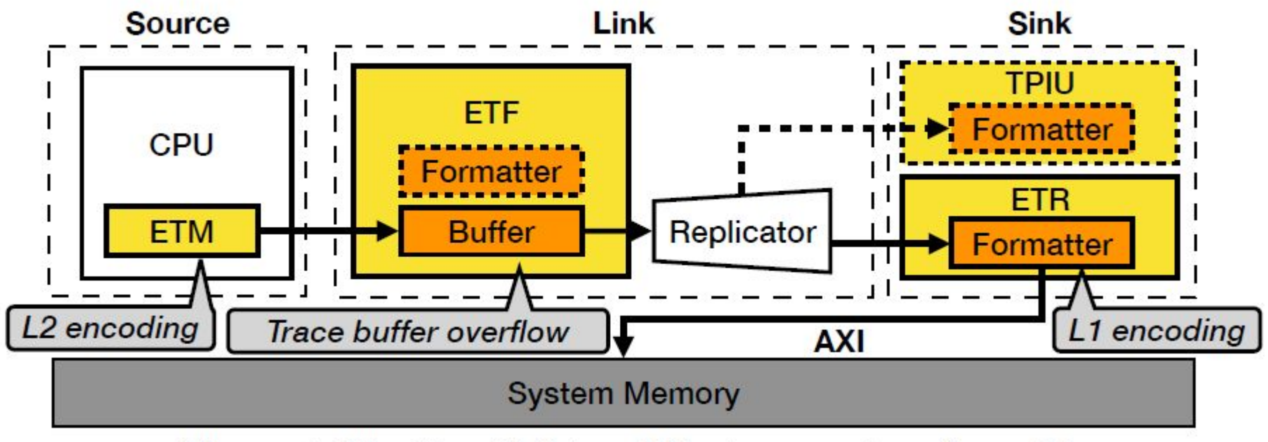
- Hardware tracing techniques (Intel PT, Arm CoreSight)
- Decoding the trace packets
- Rebuilding coverage
- Fuzzing kernel, TEE OS…



Hardware-assisted Greybox Fuzzing

Challenges in rebuilding coverage of HGF:

- **Difficult to rebuild coverage with high efficiency and moderate sensitivity**
  - Edge coverage by disassembling the binary (heavy overhead) (PTFuzz)
  - Path coverage directly from the trace packets (seed explosion) (PTrix)
  - Branch coverage by hardware features (additional decoding overhead) (μAFL)
- **Affected by the hardware tracing buffer overflow**
  - Trace data loss (imprecision and instability in coverage)



The classic architecture of CoreSight

Stalker

An efficient HGF tool based on Arm CoreSight.

Targets:

- Building moderate (branch-level) coverage with low overhead to avoid the seed explosion
- Alleviating the trace buffer overflow

Methods:

- Double-layer coverage mechanism
- Adaptive CPU frequency modulation mechanism (ACFMM)

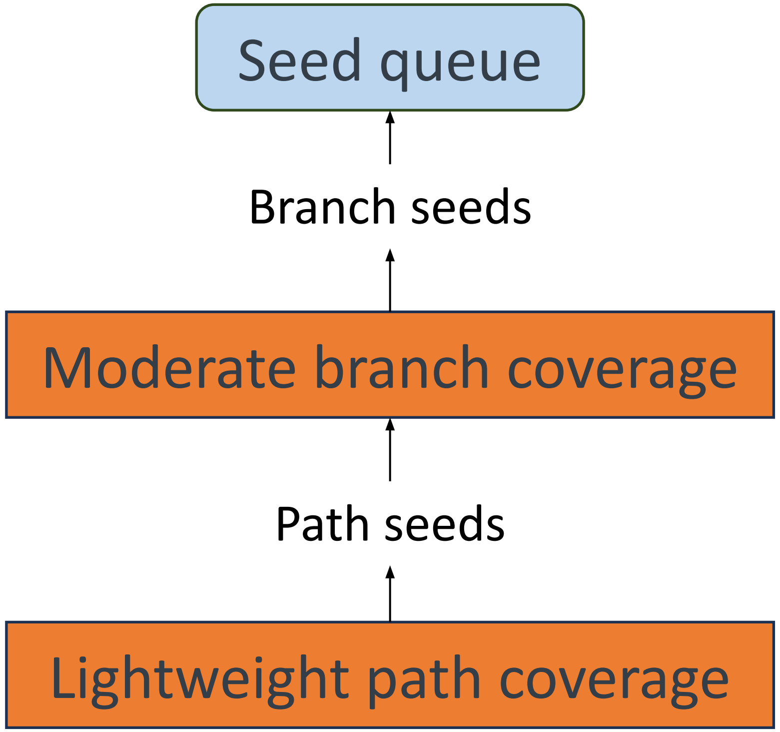
Double-layer Coverage Mechanism

Stalker utilizes this mechanism to efficiently rebuild coverage and select the seeds in moderate coverage.

Key point: assigning the execution of test cases and the selection of seeds to different coverages.

Details:

- Lightweight path coverage in bottom layer
- Moderate branch coverage in top layer



Double-layer Coverage Mechanism

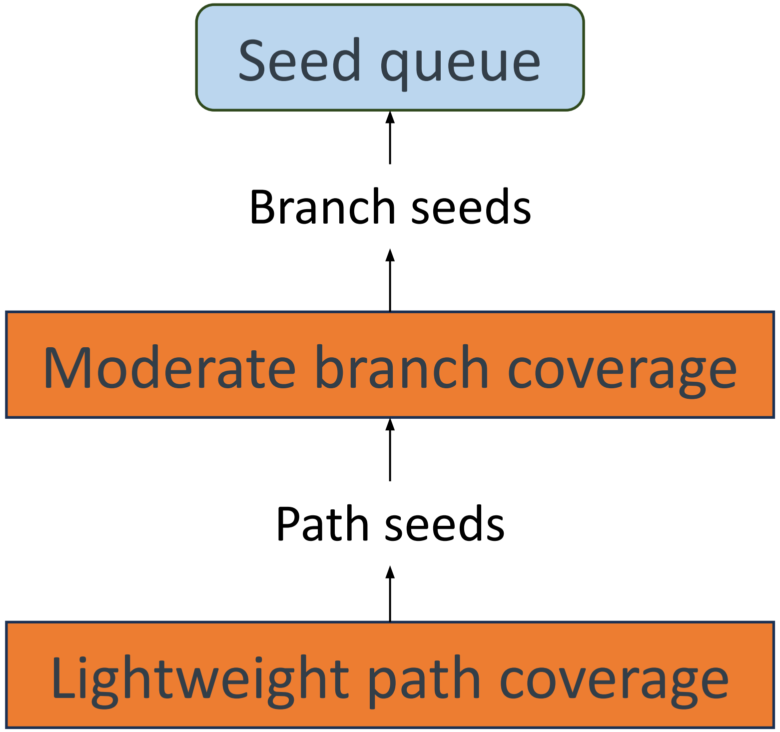
Stalker rebuilds these coverages by directly decoding the trace packets without disassembling the binaries.

Lightweight path coverage: efficiently executing the testcases and selecting the path seeds.

- ETM default mode
- Sensitive coverage (seed explosion)

Moderate branch coverage: effectively filtering the path seeds and selecting and adding the branch seeds into the seed queue.

- ETM Branch Broadcasting (BB) mode
- Additional decoding overhead



Double-layer Coverage Mechanism

Stalker also implements many strategies for efficiently and stably rebuilding coverage.

Branchless design: accelerate rebuilding coverage

Filtering noisy packets: keeping the stability of coverage

Disable formatter: reducing the decoding overhead

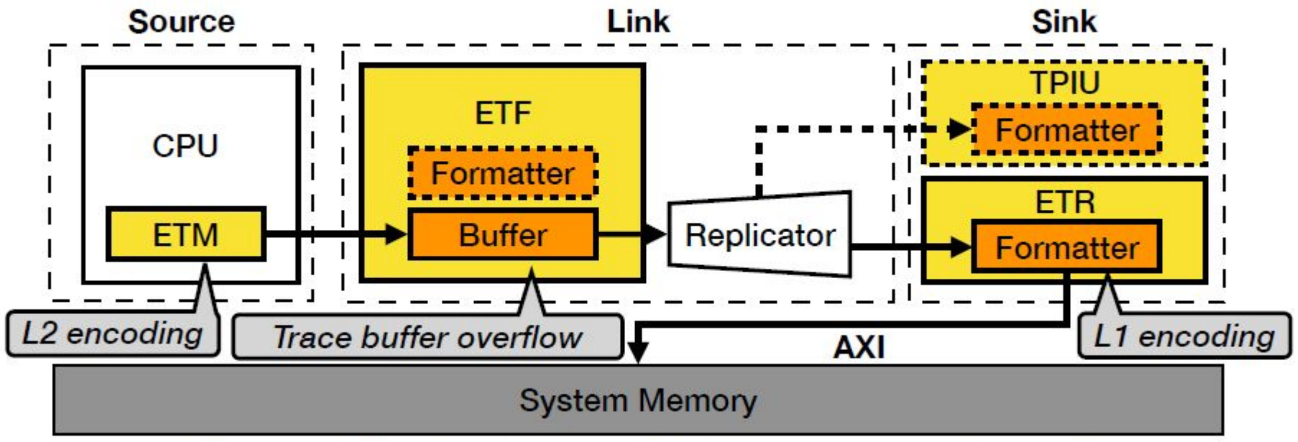
Adaptive CPU Frequency Modulation Mechanism

Stalker utilizes the ACFMM to maintain a high CPU frequency and alleviate the trace buffer overflow.

Key observation: slowing down the CPU frequency can decrease the ETM bandwidth

Key point:

- If there is frequent buffer overflow over time, reducing the CPU frequency to prevent overflow
- When no overflow occurs for an extended period, the frequency is increased to accelerate execution



The classic architecture of CoreSight.

Adaptive CPU Frequency Modulation Mechanism

Stalker utilizes the ACFMM to maintain a high CPU frequency and alleviate the trace buffer overflow.

Key observation: slowing down the CPU frequency can decrease the ETM bandwidth

Key point:

- If there is frequent buffer overflow over time, reducing the CPU frequency to prevent overflow
- When no overflow occurs for an extended period, the frequency is increased to accelerate execution

Algorithm 4 ACFMM

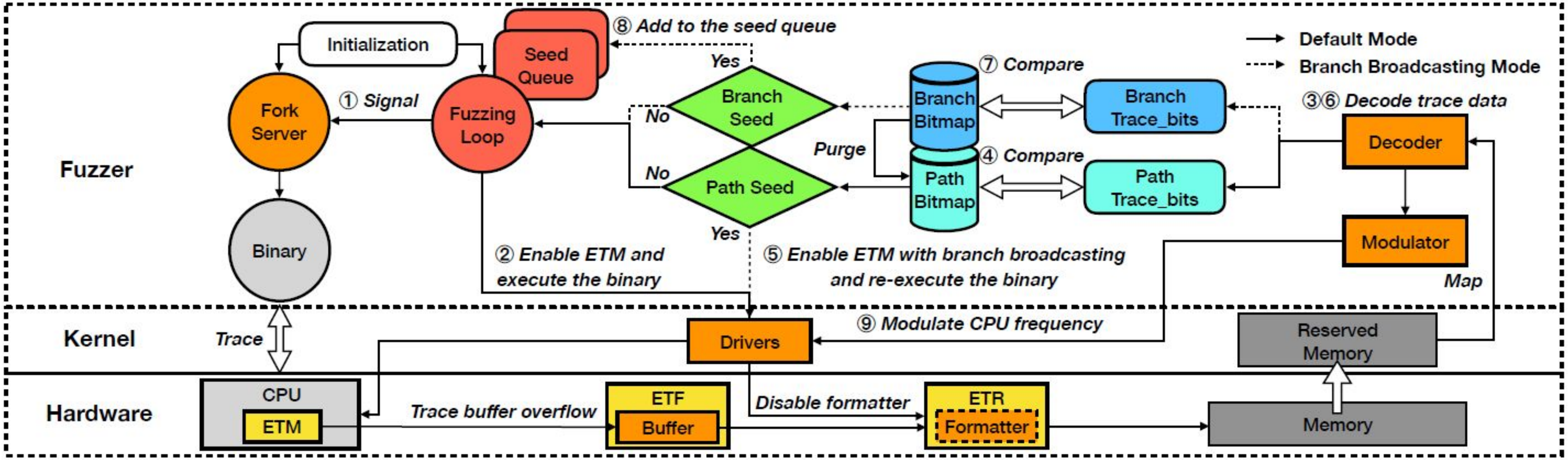
```
repeat
    testcase = Mutation(seed)
    overflow_flag = RunTarget(COV, testcase)
    if COV == PATH_COV then
        path_execs += 1
    end if
    if overflow_flag == TRUE then
        if COV == PATH_COV then
            overflow_nums_path += 1
        end if
        no_overflow_num = 0
        Decrease(cur_cpu_freq_mode)
    else
        no_overflow_num += 1
        if no_overflow_num == INTERVAL then
            Increase(cur_cpu_freq_cov)
            no_overflow_num = 0
        end if
    end if
    INTERVAL = (path_execs / overflow_nums_path) / 5
    Limit(INTERVAL)
until fuzzer exit
```

The algorithm of ACFMM.

Stalker

Stalker is built on AFL and Arm CoreSight.

- Implementing the forkserver
- Optimizing the CoreSight driver and decoder



The architecture and workflow of Stalker

Experiments Setup

Platform: Arm Juno R2 development board
(A72*2 0.6-1.2GHz, A53*4 0.45-0.95GHz)

Compare tools:

- QEMU-based: AFL-QEMU, AFL-QEMU++[1]
- CoreSight-based: Armored-CoreSight, μAFL
- PT-based: PTrix, libxdc (kAFL)

Tested software:

- 10 real-world programs selected from other papers in top conferences
- Benchmark of libxdc

Table 1: Target binaries evaluated in our evaluation.

Program	Version	Size	Format
objdump –dwarf-check -C -g -f -dwarf -x @@	binutils-v2.37	11MB	elf
readelf -a @@	binutils-v2.37	4MB	elf
nm-new -C @@	binutils-v2.37	5.8MB	elf
bsdtar -xf @@ /dev/null	libarchive-3.5.2	3.4MB	tar
nasm -f elf -o sample @@	nasm-2.15.05	3.0MB	text
bison @@	bison-3.8	2.6MB	text
tiff2bw @@ /dev/null	tiff-4.3.0	1.5MB	tiff
tiffinfo @@	tiff-4.3.0	1.6MB	tiff
xmllint @@	libxml2-2.9.10	100KB	xml
tic @@	ncurses-6.3	260KB	text

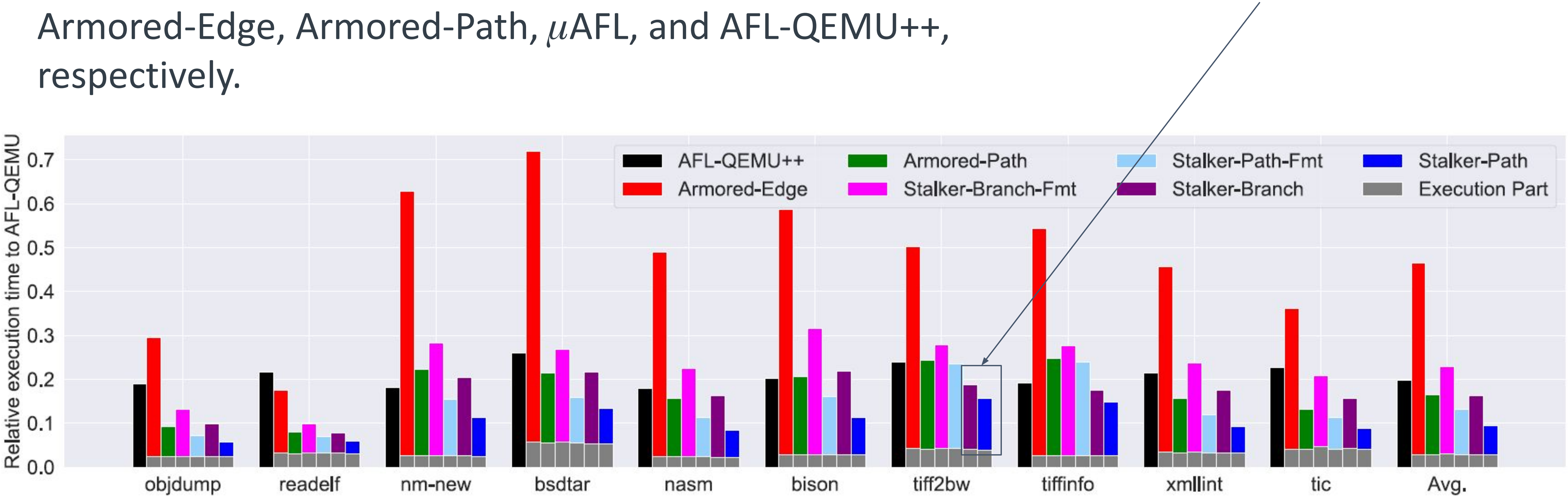
The real-world programs in our evaluation

[1] https://abiondo.me/2018/09/21/improving-afl-qemu-mode/

Efficiency of Rebuilding Coverage

Compared with CoreSight-based and QEMU-based tools:
Stalker rebuilds the same granularity coverage more efficiently, with 2.81×, 1.74×, 1.4×, and 1.22× faster than Armored-Edge, Armored-Path, μAFL, and AFL-QEMU++, respectively.

Branch coverage and path coverage of Stalker

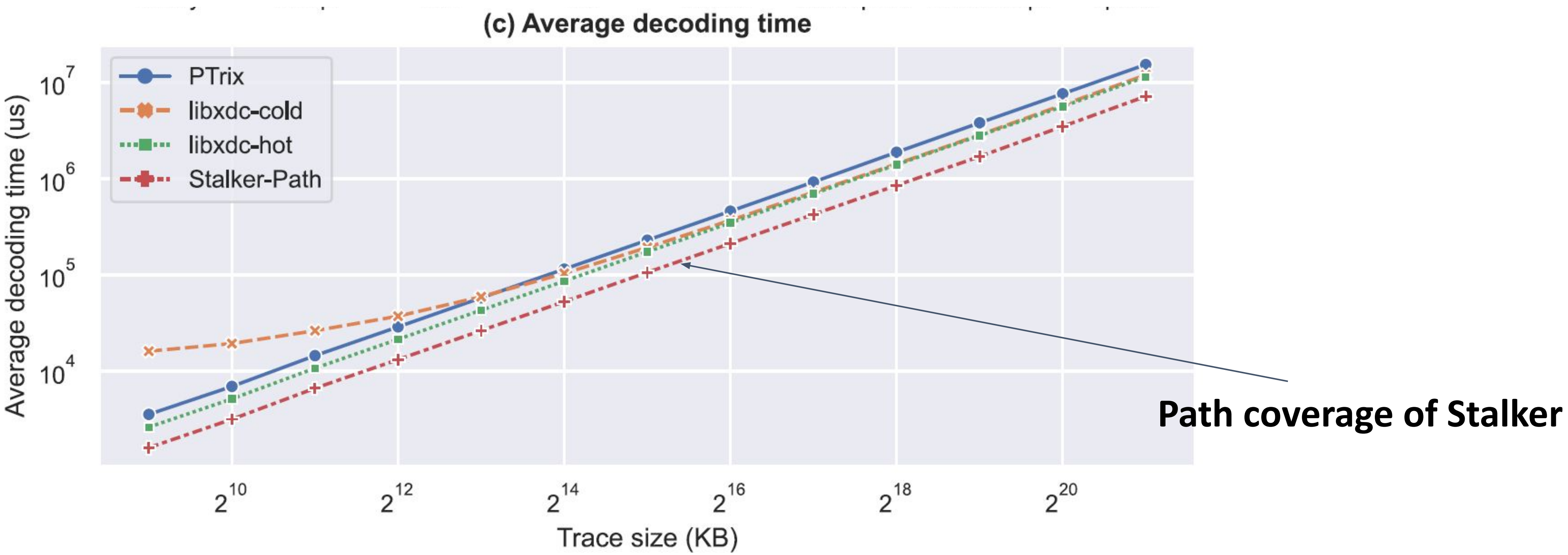


Normalized execution time of all tools with AFL-QEMU as the baseline

Efficiency of Rebuilding Coverage

Compared with PT-based tools:

Stalker-Path outperforms PTrix and libxdc by 2.2× and 1.63×, respectively.

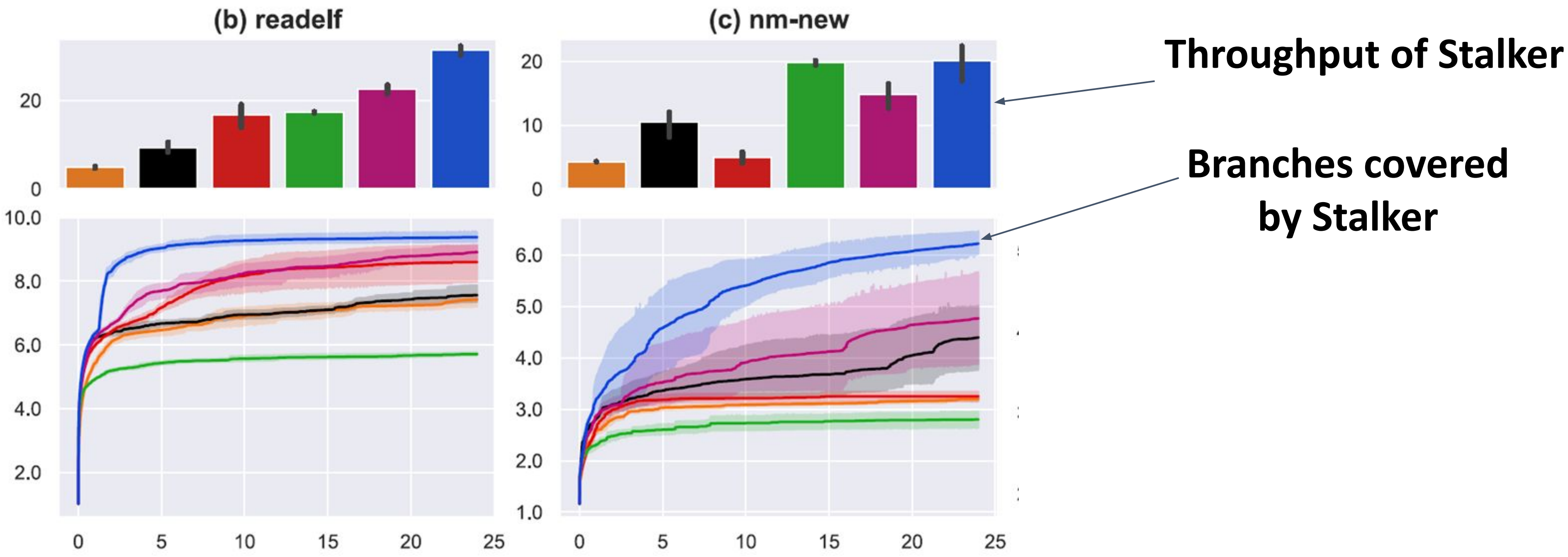


Decoding time of PTrix, libxdc-cold, libxdc-hot, and Stalker-Path on different trace file sizes

Performance of Stalker

Results:

On 10 programs, Stalker surpasses AFL-QEMU, AFL-QEMU++, Armored-Edge, Armored-Path, and μAFL with higher throughput and covering 55.7%, 13.7%, 66.4%, 323.3%, and 19.2% more paths and 21.1%, 10.0%, 23.9%, 66.1%, and 3.5% branches, respectively.



Throughput and branch coverage of six tools over 24 hours. The blue line represents Stalker.

Double-Layer Coverage Mechanism

Metrics: counting the number of seeds before and after the filtering as N_q and N_f, defined the sensitive ratio as N_q/N_f.

Results:

- Improving the speed of Stalker as executing 86.7% test cases with lightweight coverage
- Filtering 6,124 branch seeds from the 1.62M path seeds and adding them into the queue
- The sensitivity of branch coverage in Stalker is comparable to that of the branch-count coverage

Path seeds   Branch seeds

Table 3: Numbers of selected and filtered seeds of the hardware-assisted tools, respectively. The right column denotes the number of seeds filtered by AFL in the branch-count coverage. The two left columns in STALKER, and one in others represent the number of seeds selected by them.

Binary	Armored-Edge	Armored-Path	STALKER-Branch-Fmt	STALKER (Path Seed/Branch Seed)
objdump	8,820/5,277	51.55K/2,199	9,850/7,627	0.42M/10.63K/8,476
readelf	9,271/6,358	67.05K/1,715	9,335/6,639	1.54M/10.4K/7,345
nm-new	3,202/1,058	56.24K/591	3,781/2,478	0.35M/7,576/4,850
bsdtar	5,750/1,701	64.91K/193	3,034/1,879	4.75M/3,225/2,043
nasm	4,156/1,706	83.29K/1,279	7,324/4,477	0.48M/8,164/4,889
bison	3,367/805	66.16K/814	3,388/2,197	1.11M/3,627/2,282
tiff2bw	2,924/1,377	60.17K/547	2,371/1,717	2.12M/2,566/1,842
tiffinfo	4,135/2,152	59.86K/626	3,353/2,295	1.93M/3,513/2,422
xmllint	6,641/2,598	0.2M/1,192	5,221/3,021	2.18M/5,509/3,115
tic	5,164/2,163	0.26M/753	5,330/2,843	1.28M/6,032/3,118
Avg.	5,343/2,519	97.37K/990	5,298/3,517	1.62M/6,124/4,038
N_q/N_f	212%	9835%	151%	26392%/152%

The number of seeds before and after the filtering.

Sensitive ratio

Adaptive CPU Frequency Modulation Mechanism

Effectiveness of ACFMM:

- Achieving the highest coverage with ACFMM
- Reducing the overflow percentage from 40.92% to 2.46%

Table 5: Average branches, number of branch and path seeds (B/P seeds), throughput, and overflows of STALKER, STALKER-A72-0.6GHz, and STALKER-A72-1.2GHz on nasm. Numbers in the brackets of Throughput: percentages of the test cases executed under 0.6GHz, 1.0GHz, and 1.2GHz.

Config	Branch	B/P Seeds	Throughput(0.6/1.0/1.2)	Overflow
ACFMM	5,649	9,202/890K	3.93M(58%/22%/21%)	96,909(2.46%)
A72-0.6GHz	5,435	8,465/507K	2.23M(100%/-/-)	0.23M(10.34%)
A72-1.2GHz	5,188	22,073/1.17M	4.12M(-/-/100%)	1.68M(40.92%)

The results of disabling/enabling the ACFMM.

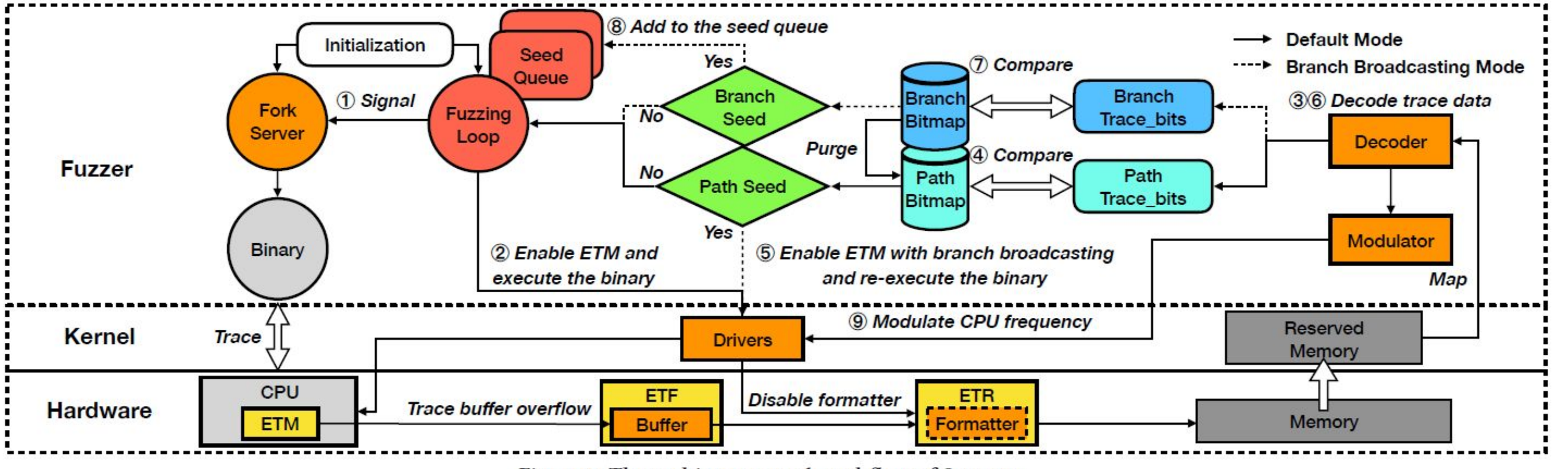
Branches covered by Stalker

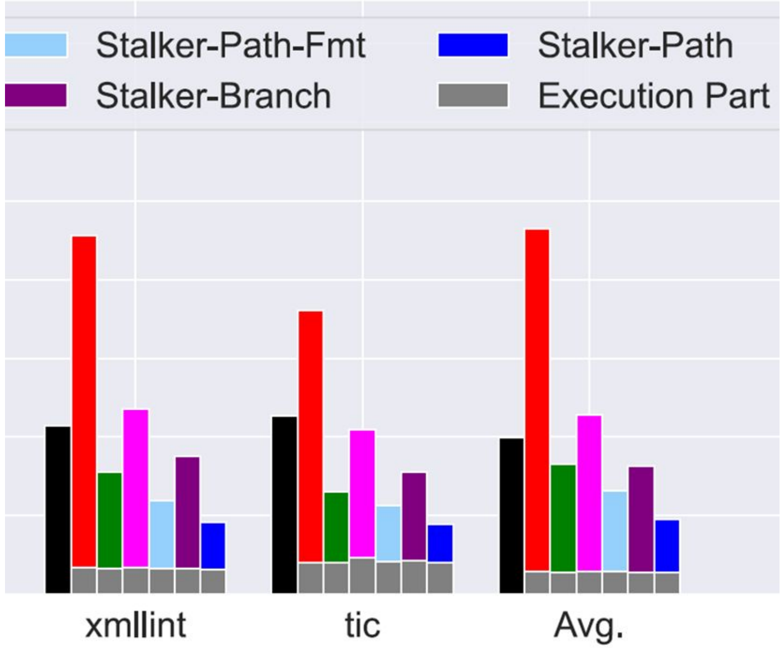
Percentage of overflow testcases

Conclusion

Stalker: an efficient hardware-assisted greybox fuzzer based on Arm CoreSight.

- Double-layer coverage mechanism
- Adaptive CPU frequency modulation mechanism





yuetai17@nudt.edu.cn