Google

Klee in FuzzBench

Laurent Simon, Read Sprabery, Jonathan Metzman

@FuzzBench Team

June 10, 2021

What's FuzzBench

GitHub: https://github.com/google/fuzzbench

Need to integrate new fuzzing techniques

- Lots of research
- Expensive to
  - Re-implement
  - Port proof-of-concept
  - Deploy at scale (e.g. OSS-Fuzz, free service for open source fuzzing)
- Need prioritization

GitHub: https://github.com/google/fuzzbench
https://github.com/google/oss-fuzz

No standard evaluation framework

- Experimental design
  - Number of trials, time of trials
  - Program selection
  - Fuzzer configurations (seeds, dictionaries, flags, etc)
- Result reporting
  - Statistical significance
  - Plotting library
  - Coverage (blocks, lines, edges, etc)
- Scaling: native cloud support

GitHub: https://github.com/google/fuzzbench

Open source fuzzing benchmarking platform

- FuzzBench
  - ~20 real-world programs from OSS-Fuzz
  - Runs
  - Extract coverage
  - De-duplicate bugs
  - Plot results
  - Report statistical significance
- Complements micro-benchmarks
  - Competition on Software Testing[1]
  - coreutils

[1] **GitHub:** https://test-comp.sosy-lab.org/2021/

Integration Klee with FuzzBench

Benchmarks

- <10 benchmarks out of ~20 available in Fuzzbench
- ~400 LoC of Python code
- Few weeks on and off
- Patched Klee to generate binary files instead of .ktest files
    - Can be replayed to extract coverage information

Unsupported instructions

- Vectorized, floating points
  - Vorbis:
    - LLVM ERROR: Code generator does not support intrinsic function 'llvm.rint.f64'!
  - Libpng gives error
    - silently concretizing (reason: floating point)
- Longjump
  - Libpng
    - KLEE: ERROR: ... /illegal.c:40: longjmp unsupported
- Assembly instructions

Unsupported libraries and dependencies

- Pthread support
  - KLEE: WARNING: undefined reference to function: pthread_create

- System-level linking
  - Many programs link against zlib, openssl, etc
    - Need to recursively compile these or tell KLEE to ignore them
    - Often these are in parts of code we don't need during fuzzing anyway

OOM'ing

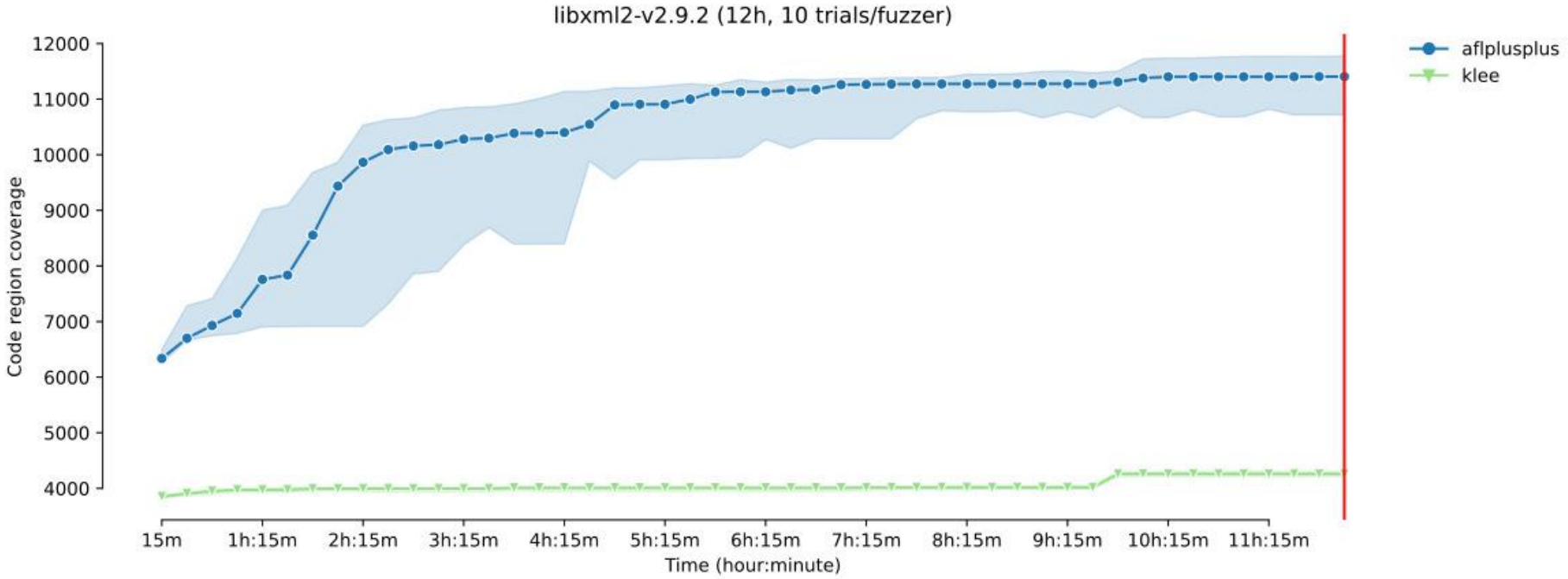
- Klee cannot reliably respect the memory usage specified via -max-memory option

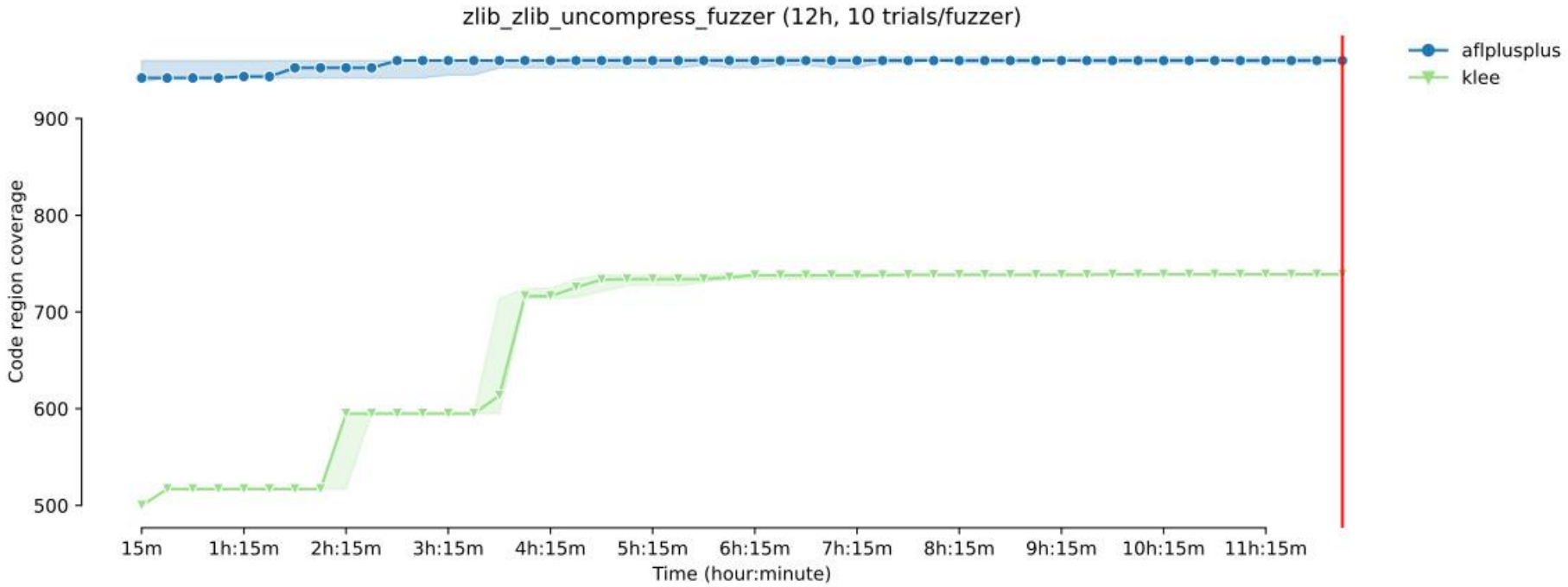
Experiments & Results

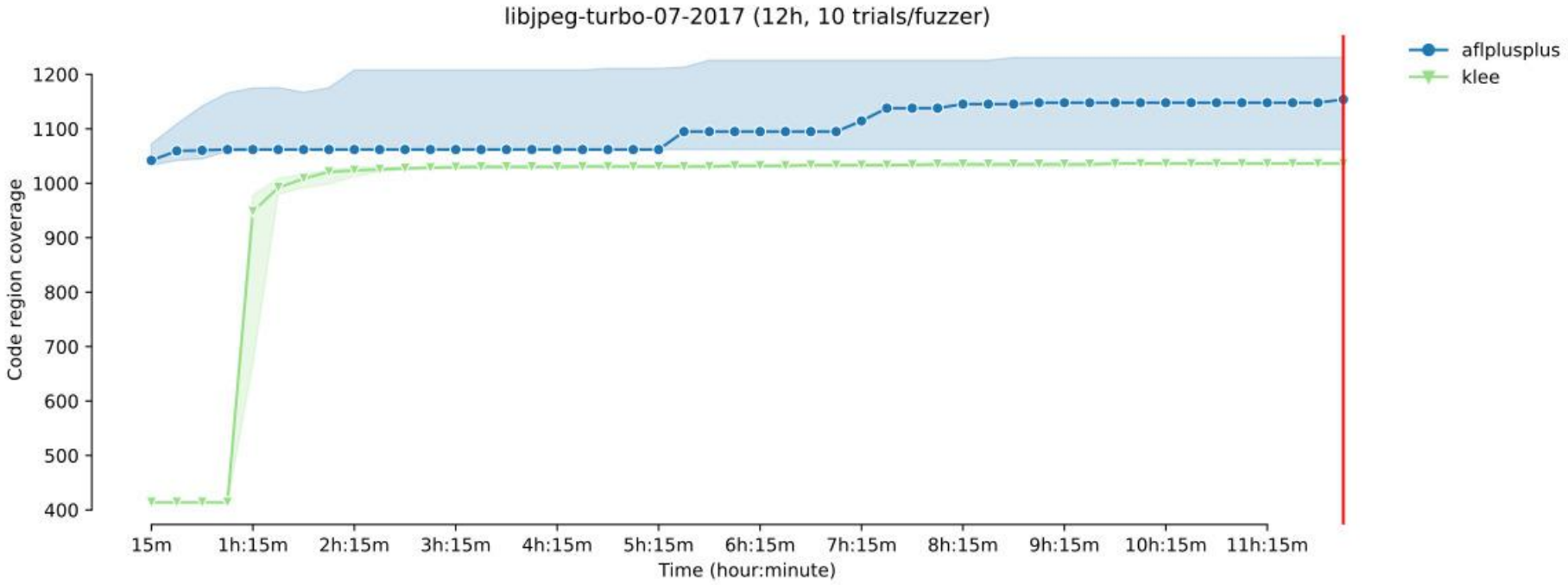
Setup 1: no seeds

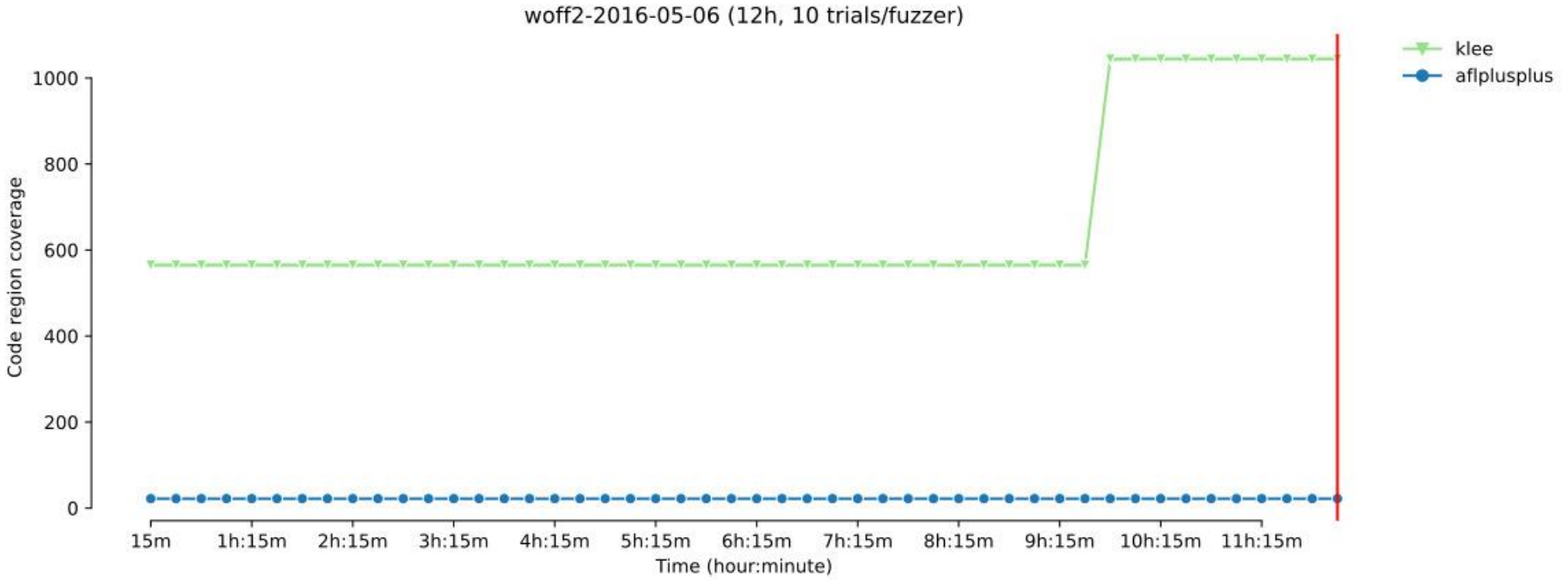
- Afl++ vs Klee (-max-memory=3GB)
- **No seed corpus**
- No dictionary
- 10 runs
- 12 hrs

GitHub: https://github.com/AFLplusplus/AFLplusplus


libxml2-v2.9.2 (12h, 10 trials/fuzzer)
Code region coverage
12000
11000
10000
9000
8000
7000
6000
5000
4000
15m
1h:15m
2h:15m
3h:15m
4h:15m
5h:15m
6h:15m
7h:15m
8h:15m
9h:15m
10h:15m
11h:15m
Time (hour:minute)
aflplusplus
klee


zlib_zlib_uncompress_fuzzer (12h, 10 trials/fuzzer)
aflplusplus
klee
Code region coverage
900
800
700
600
500
15m
1h:15m
2h:15m
3h:15m
4h:15m
5h:15m
6h:15m
7h:15m
8h:15m
9h:15m
10h:15m
11h:15m
Time (hour:minute)


libjpeg-turbo-07-2017 (12h, 10 trials/fuzzer)
aflplusplus
klee
Code region coverage
1200
1100
1000
900
800
700
600
500
400
15m
1h:15m
2h:15m
3h:15m
4h:15m
5h:15m
6h:15m
7h:15m
8h:15m
9h:15m
10h:15m
11h:15m
Time (hour:minute)


woff2-2016-05-06 (12h, 10 trials/fuzzer)
klee
aflplusplus
Code region coverage
0
200
400
600
800
1000
15m
1h:15m
2h:15m
3h:15m
4h:15m
5h:15m
6h:15m
7h:15m
8h:15m
9h:15m
10h:15m
11h:15m
Time (hour:minute)

Setup 1: takeaways

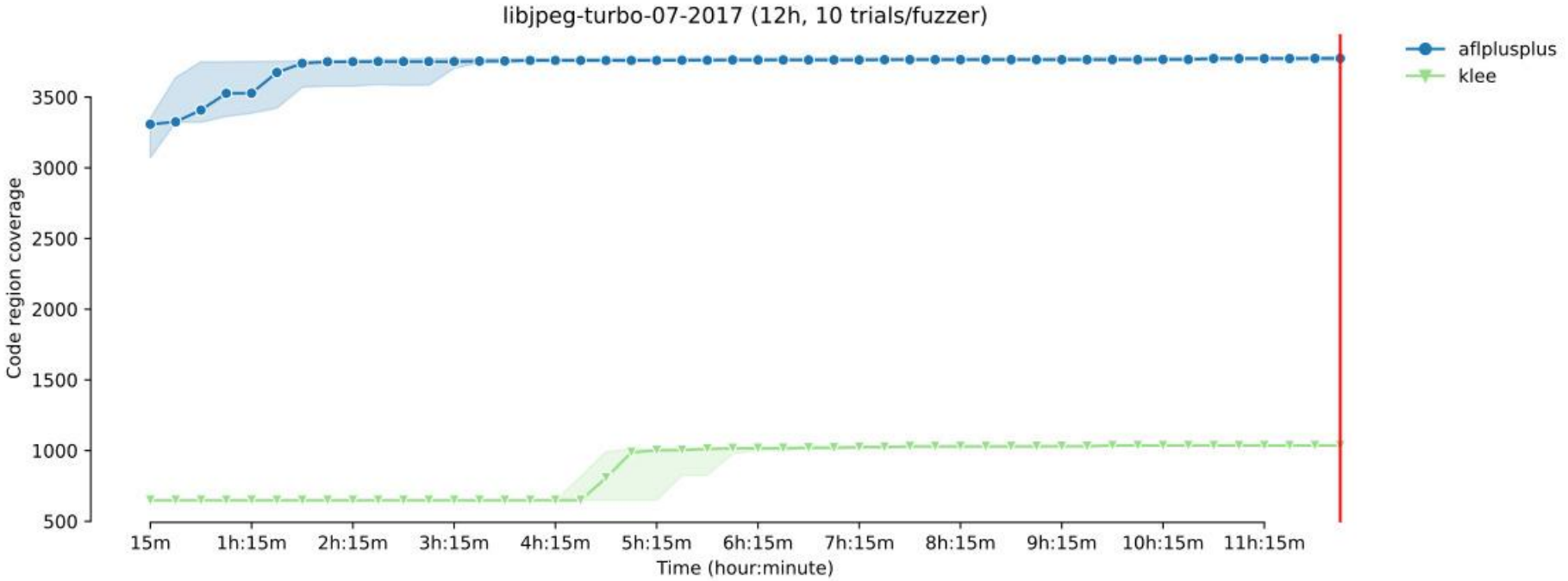
- General trend
  - AFL++ visits more code than Klee, 2x to 3x
  - AFL++ visits all the code that Klee visits

- On woff2 target, Klee visits ~1000 LoC
  - AFL++ is stuck at ~80 LoC

- Klee is more consistent than AFL++
  - Less variance

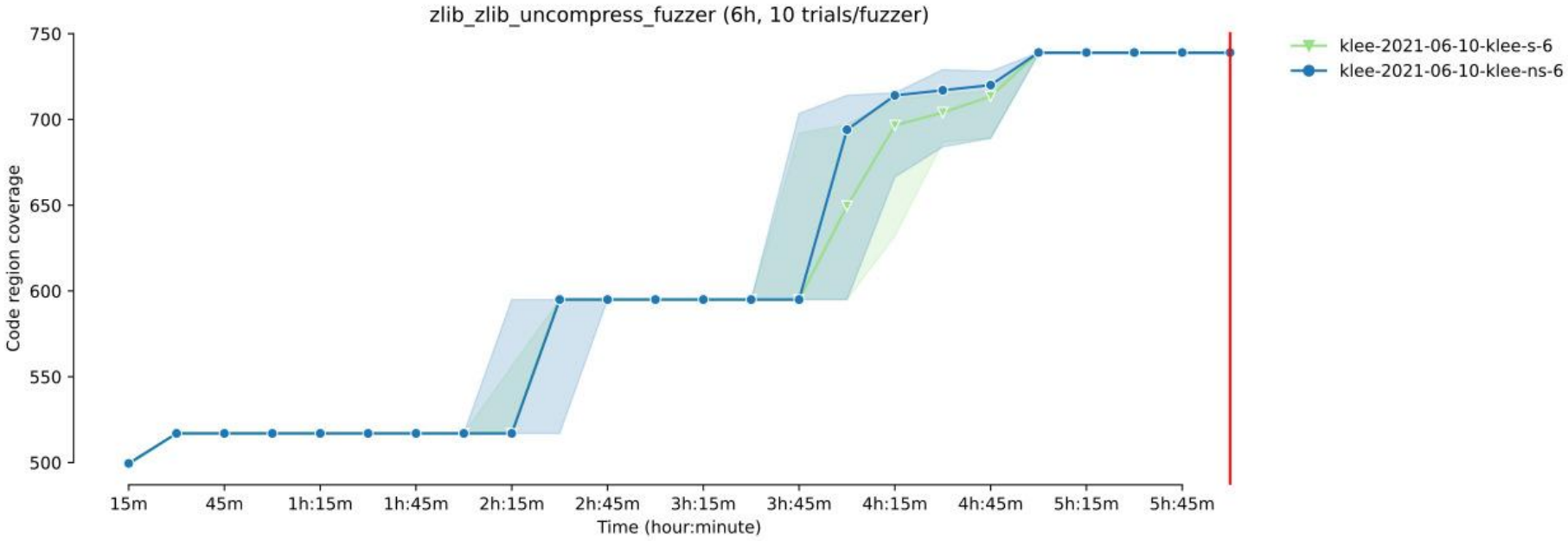
Setup 2: with seeds

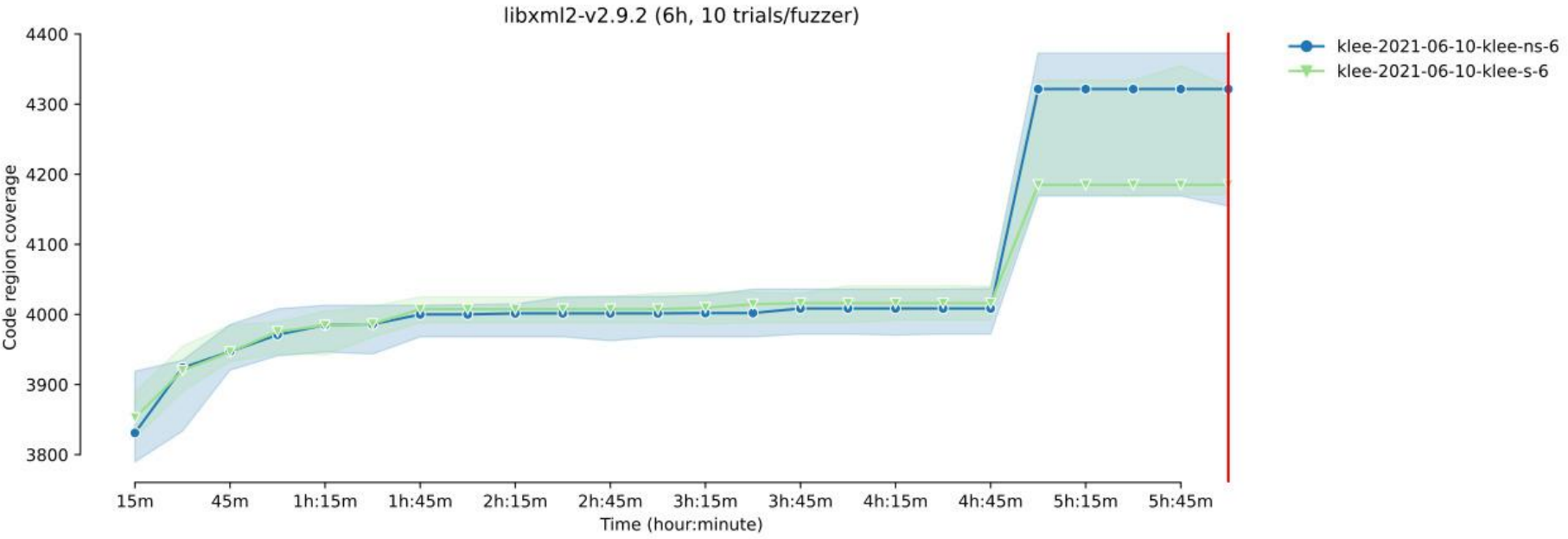
- Afl++ vs Klee (-max-memory=3GB)
- **Seed corpus** (not saturated, from OSS-Fuzz)
- No dictionary
- 10 runs
- 12 hrs


libjpeg-turbo-07-2017 (12h, 10 trials/fuzzer)
aflplusplus
klee
Code region coverage
3500
3000
2500
2000
1500
1000
500
15m
1h:15m
2h:15m
3h:15m
4h:15m
5h:15m
6h:15m
7h:15m
8h:15m
9h:15m
10h:15m
11h:15m
Time (hour:minute)

Setup 2: takeaways

- AFL++ makes good use of seeds
- Seeds have little impact on Klee's performance


zlib_zlib_uncompress_fuzzer (6h, 10 trials/fuzzer)
Code region coverage
750
700
650
600
550
500
15m
45m
1h:15m
1h:45m
2h:15m
2h:45m
3h:15m
3h:45m
4h:15m
4h:45m
5h:15m
5h:45m
Time (hour:minute)
klee-2021-06-10-klee-s-6
klee-2021-06-10-klee-ns-6


libxml2-v2.9.2 (6h, 10 trials/fuzzer)
Code region coverage
4400
4300
4200
4100
4000
3900
3800
15m
45m
1h:15m
1h:45m
2h:15m
2h:45m
3h:15m
3h:45m
4h:15m
4h:45m
5h:15m
5h:45m
Time (hour:minute)
klee-2021-06-10-klee-ns-6
klee-2021-06-10-klee-s-6

Going forward

Blockers for adoption of symex in industry

- Engineering blockers
  - Run klee at scale to find bugs and for research evaluation
  - Instruction support, compilation toolchain, asm, etc

- Fuzzers 's ease of use
- Coverage
  - State explosion
  - Klee does not visit new 'code' after a few hours

Sweet spot for adoption of symex in industry

- OSS-Fuzz
  - Run 24/7 for 365 days/year
  - Coverage/bugs improvement marginal after 24hrs
- Goal:
  - Target **smaller**, **unexplored code** by fuzzers
    - Reduce state explosion
    - Take advantage of SMT solvers
  - Examples:
    - Concolic execution, e.g. follow a path and start exploring symbolically
    - MoKlee https://srg.doc.ic.ac.uk/projects/moklee/

How can Google help?

- Fuzzbench, OSS-Fuzz are frees service you can use

- We have funding for
  - Research grants
  - Integration of symex engines/fuzzers into FuzzBench for evaluation
  - 5-50K or higher based on projects

- Tell us how we can help

Thank You