OS-State–Aware Fuzzing for Worst-Case Response Times

Herbsttreffen 2022 der Fachgruppe Betriebssysteme

20. September 2022

Alwin Berger[1], Simon Schuster[2], Peter Wägemann[2] and Peter Ulbrich[1]

(1) Technische Universität Dortmund

`https://sys.cs.tu-dortmund.de/`

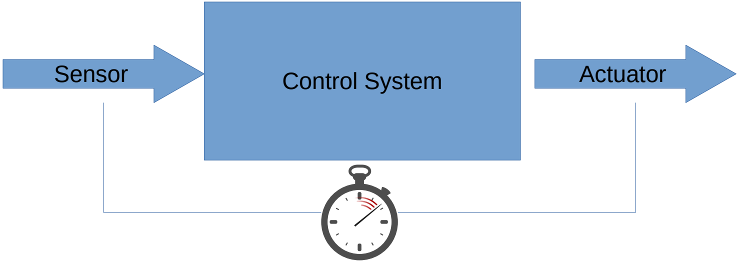
(2) Friedrich-Alexander-Universität Erlangen-Nürnberg

`https://sys.cs.fau.de/`

technische universität
dortmund

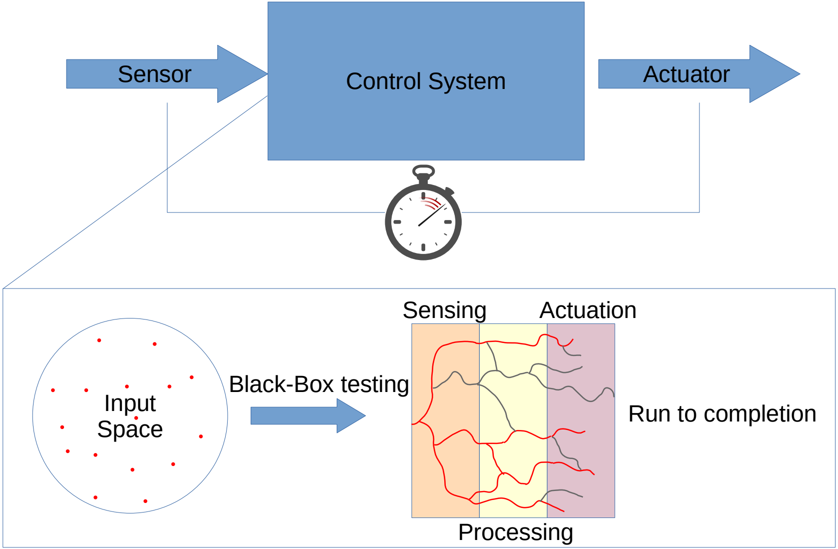
arbeitsgruppe
systemsoftware

Motivation



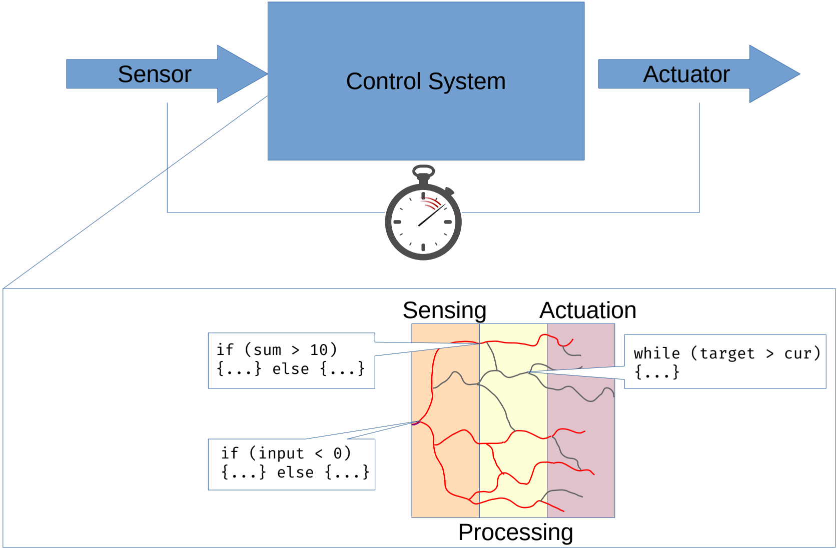
Find the End-to-End latency of a system

Motivation



Input space can not be fully covered

Motivation



Sensor

Control System

Actuator

Sensing

Actuation

Processing

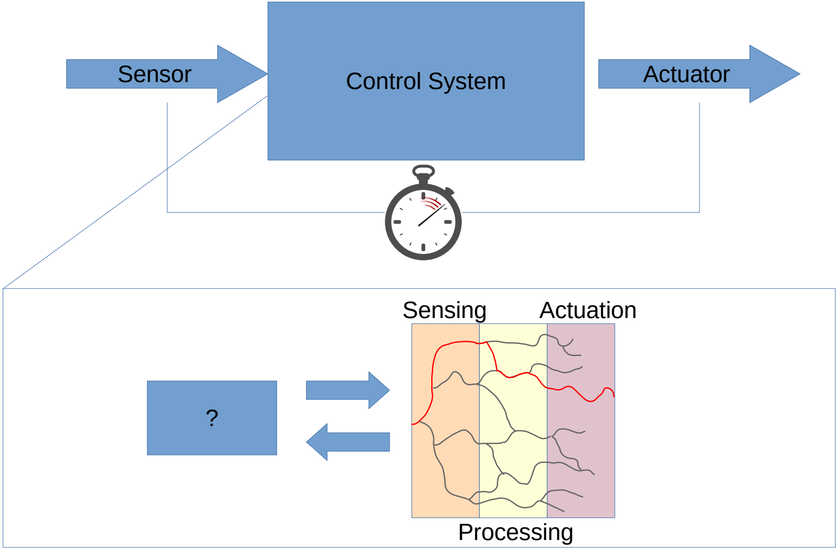
```
if (sum > 10)
{...} else {...}
```

```
while (target > cur)
{...}
```

```
if (input < 0)
{...} else {...}
```

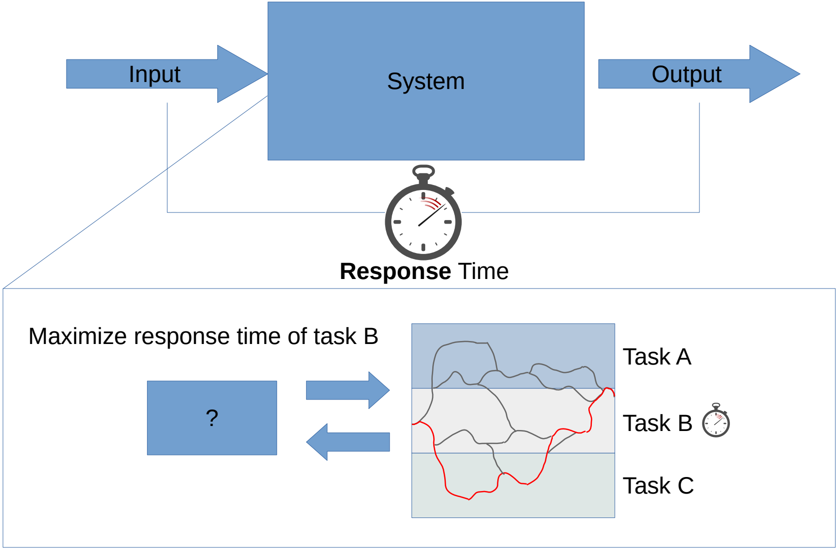
Conditions and loop iteration make certain paths hard to reach

Motivation



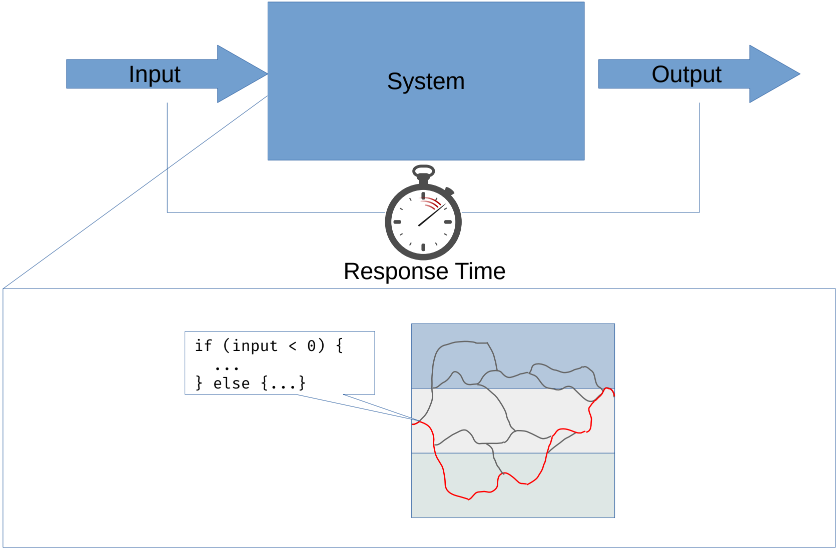
Goal: Systematically find long paths

Motivation - Response Time



Generalize to response time in preemptive multitasking system

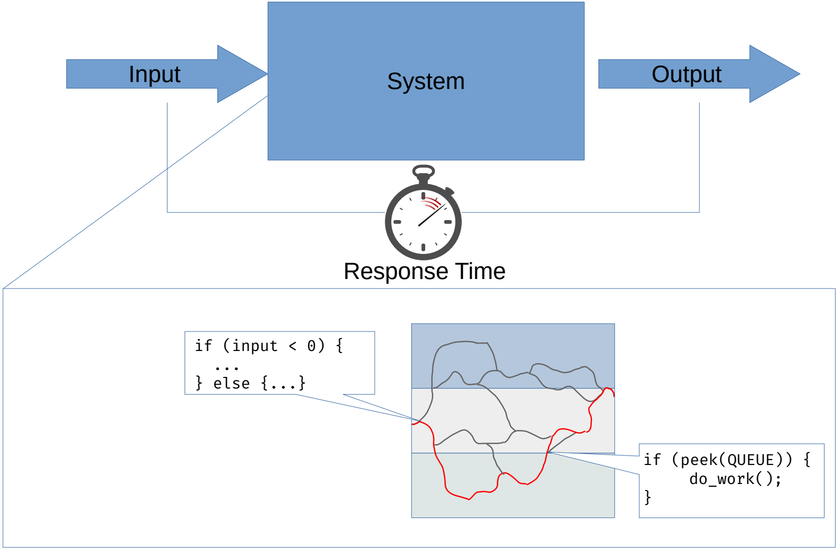
Motivation - Response Time



```
if (input < 0) {
   ...
} else {...}
```

Influences: ① inputs

Motivation - Response Time

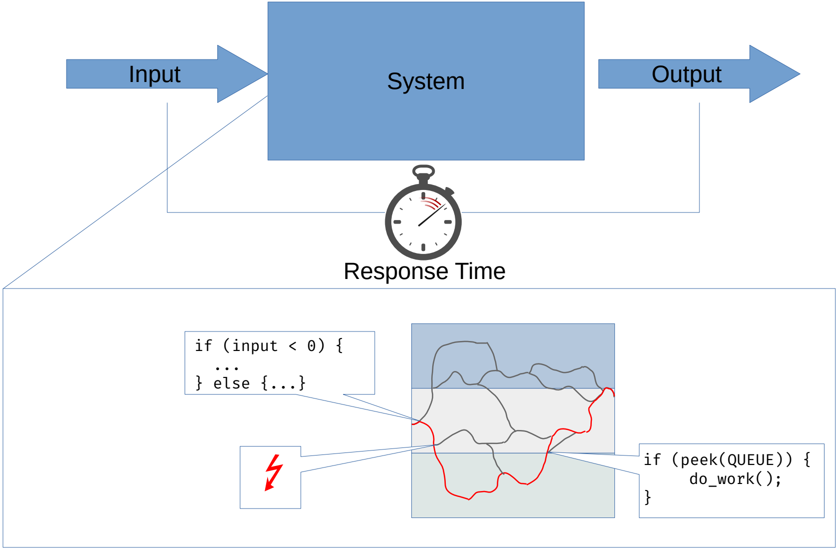


```
if (input < 0) {
  ...
} else {...}
```

```
if (peek(QUEUE)) {
    do_work();
}
```

Influences: ① inputs ② concurrent tasks

Motivation - Response Time

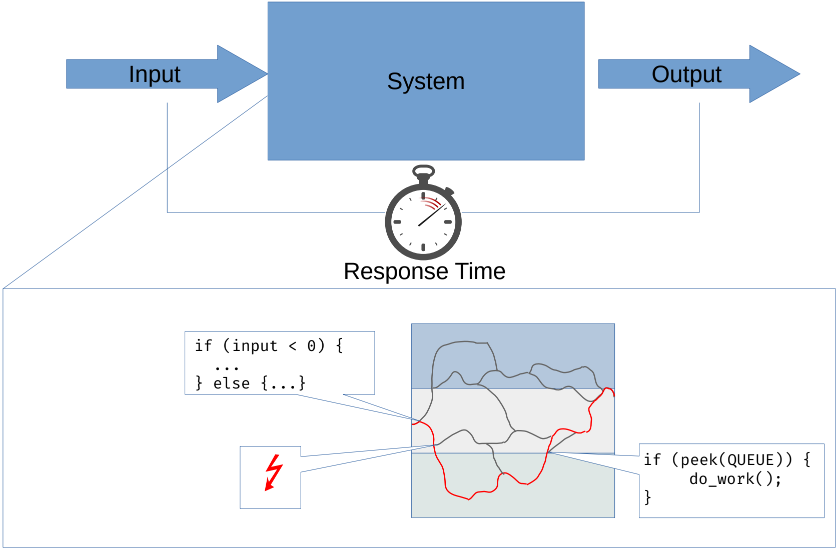


```
if (input < 0) {
   ...
} else {...}
```

```
if (peek(QUEUE)) {
    do_work();
}
```

Influences: ① inputs ② concurrent tasks ③ asynchronous events ④ ...

Motivation - Response Time



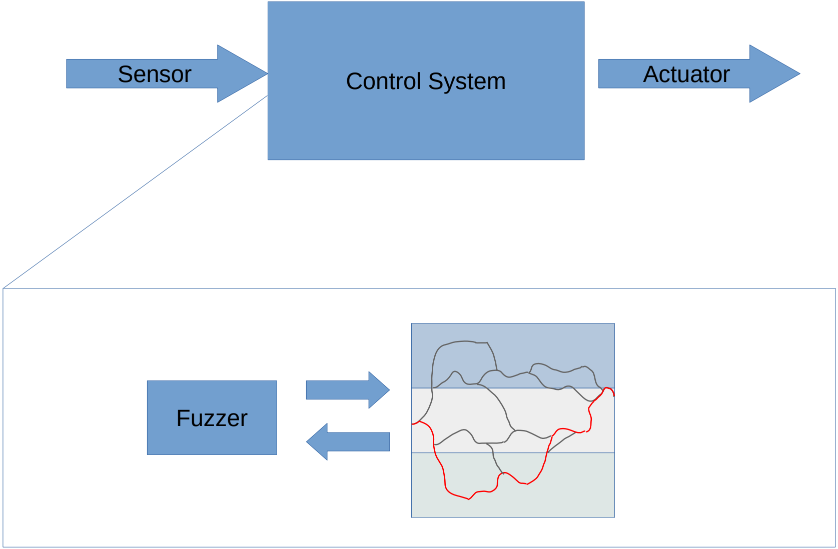
```
if (input < 0) {
   ...
} else {...}
```

```
if (peek(QUEUE)) {
    do_work();
}
```

Influences: ① inputs ② concurrent tasks ③ asynchronous events ④ ...

System state

Motivation - Fuzzing



Fuzzing uses coverage-based heuristics to explore code branches.

➔ Needs adjustments for response times

Problem Analysis

Input space too large for random testing

- 💡 Use fuzzing to explore code
  - Grey-box: track and maximize branch-coverage
  - No prior knowledge needed
  - Successful in security research

	Random	Fuzzing
Guided exploration	✗	✓

Problem Analysis 1/2

Input space too large for random testing

- 💡 Use fuzzing to explore code
  - Grey-box: track and maximize branch-coverage
  - No prior knowledge needed
  - Successful in security research
- ⚠ Challenge 1: Re-target fuzzer for temporal objective

	Random	Fuzzing
Guided exploration	✗	✓
Temporal objective	✗	✗

Problem Analysis 1/2

Input space too large for random testing

- 💡 Use fuzzing to explore code
  - Grey-box: track and maximize branch-coverage
  - No prior knowledge needed
  - Successful in security research
- ⚠ Challenge 1: Re-target fuzzer for temporal objective
- Related: PerfFuzz
  - Maximize execution counts in hotspots
  - → No response time maximization

▶ Lemieux, Padhye, Sen, Song.
PerfFuzz: Automatically Generating Pathological Inputs.
27th ACM SIGSOFT International Symposium on Software Testing and Analysis (*ISSTA '18*)

	Random	Fuzzing
Guided exploration	✗	✓
Temporal objective	✗	✓

Problem Analysis 1/2

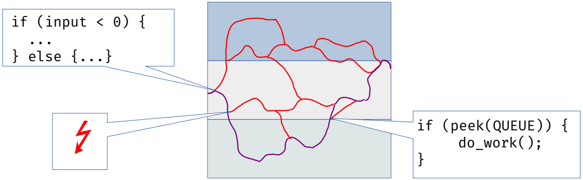
Input space too large for random testing

- 💡 Use fuzzing to explore code
  - Grey-box: track and maximize branch-coverage
  - No prior knowledge needed
  - Successful in security research
- ⚠ Challenge 1: Re-target fuzzer for temporal objective
- Related: PerfFuzz
  - Maximize execution counts in hotspots
  - → No response time maximization

▶ Lemieux, Padhye, Sen, Song.
PerfFuzz: Automatically Generating Pathological Inputs.
27th ACM SIGSOFT International Symposium on Software Testing and Analysis (*ISSTA '18*)

	Random	Fuzzing
Guided exploration	✗	✓
Temporal objective	✗	✓
Reponse maximization	✗	✗

Problem Analysis 2/2



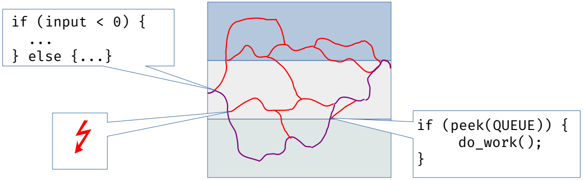
```
if (input < 0) {
  ...
} else {...}
```

```
if (peek(QUEUE)) {
     do_work();
}
```

Worst-case task execution times $\not\Rightarrow$ worst-case response time

→ System-wide control flow affects time

Problem Analysis



```
if (input < 0) {
  ...
} else {...}
```

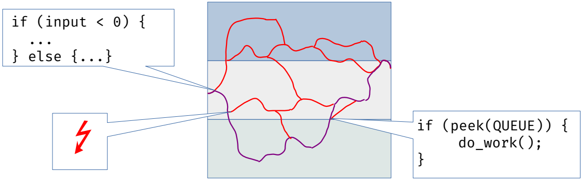
```
if (peek(QUEUE)) {
     do_work();
}
```

Worst-case task execution times $\nRightarrow$ worst-case response time

→ System-wide control flow affects time

⚠ Challenge 2: Extend task-local to system-wide control flow view

Problem Analysis



```
if (input < 0) {
  ...
} else {...}
```

```
if (peek(QUEUE)) {
     do_work();
}
```

Worst-case task execution times $\not\Rightarrow$ worst-case response time

→ System-wide control flow affects time

⚠ Challenge 2: Extend task-local to system-wide control flow view

⚠ Challenge 3: Model asynchronous events

Outline

Introduction

Approach: **F**uzzing **RE**sponse **T**imes (FRET)

FRET Implementation

Evaluation

Summary

Concept

Challenges		Our Solutions
Increase response time	→	Heuristics rewarding time increases
View system control flow		
Asynchronous events		

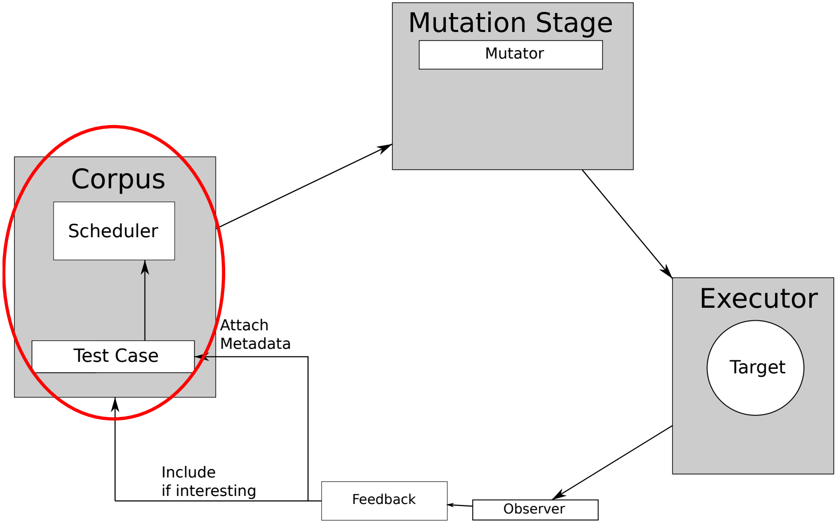
Concept

Challenges		Our Solutions
Increase response time	→	Heuristics rewarding time increases
View system control flow	→	Capture system state paths and reward improvements within
Asynchronous events		

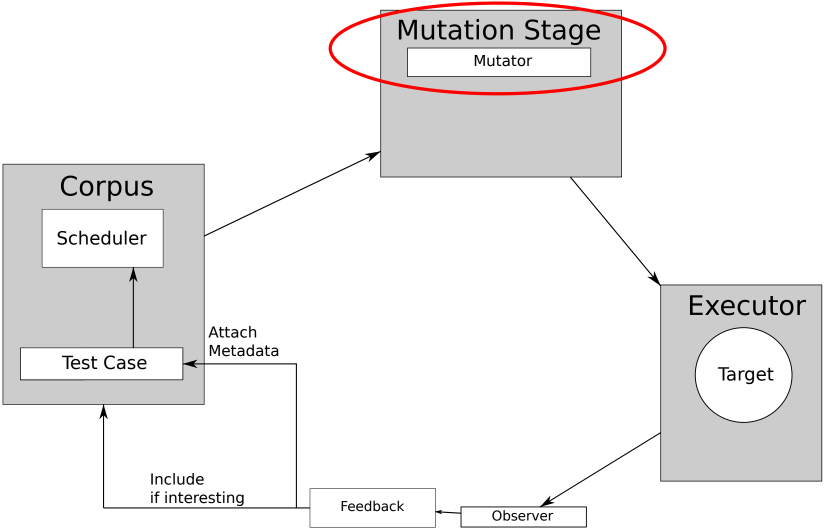
Concept

Challenges		Our Solutions
Increase response time	→	Heuristics rewarding time increases
View system control flow	→	Capture system state paths and reward improvements within
Asynchronous events	→	Model interrupt release times as fuzzer inputs

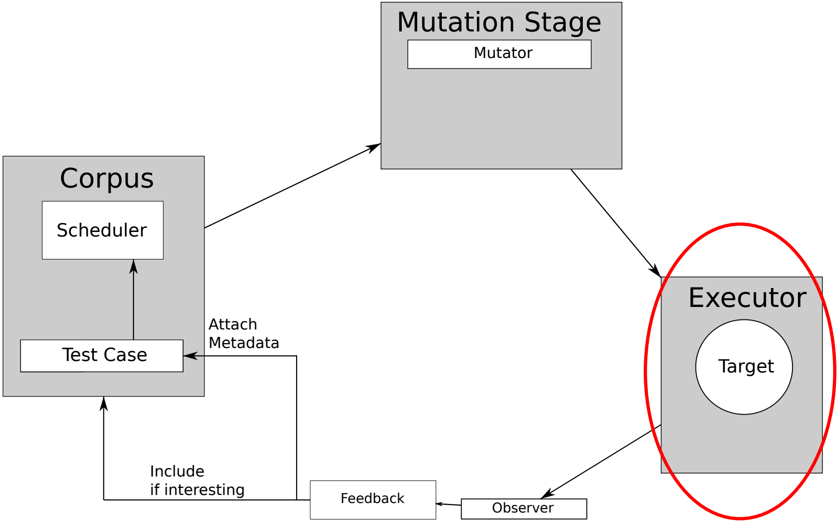
Background: Fuzzers



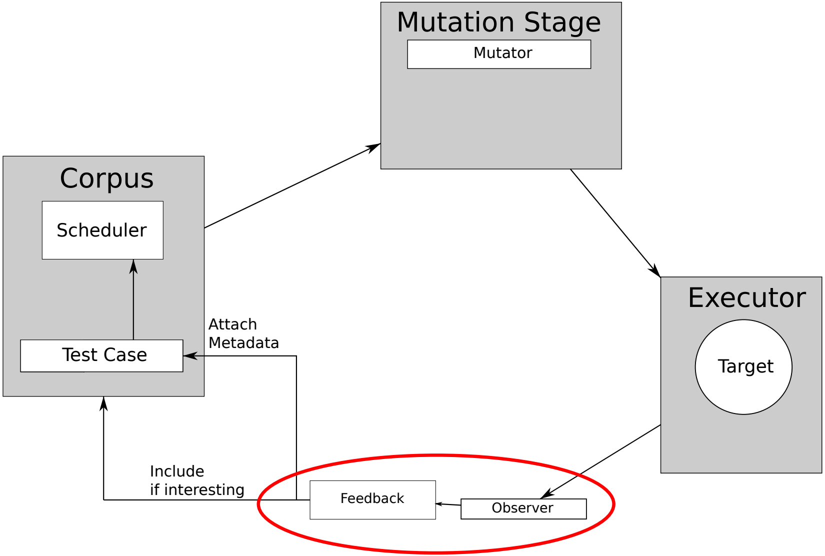
Background: Fuzzers



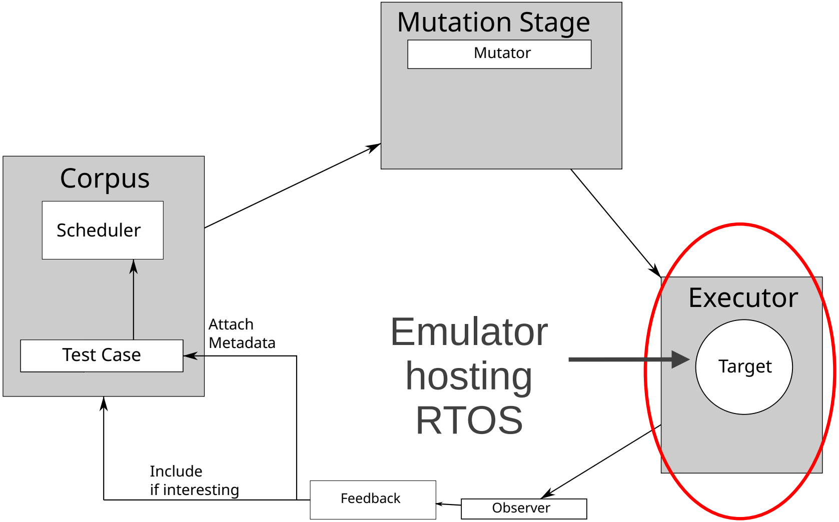
Background: Fuzzers



Background: Fuzzers

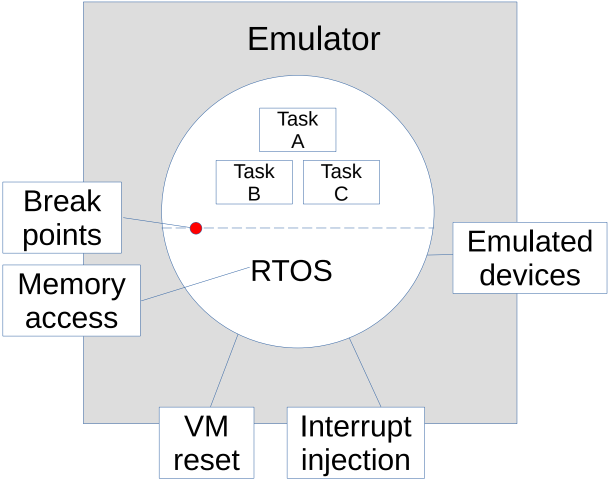


WCRT Fuzzers - Execution



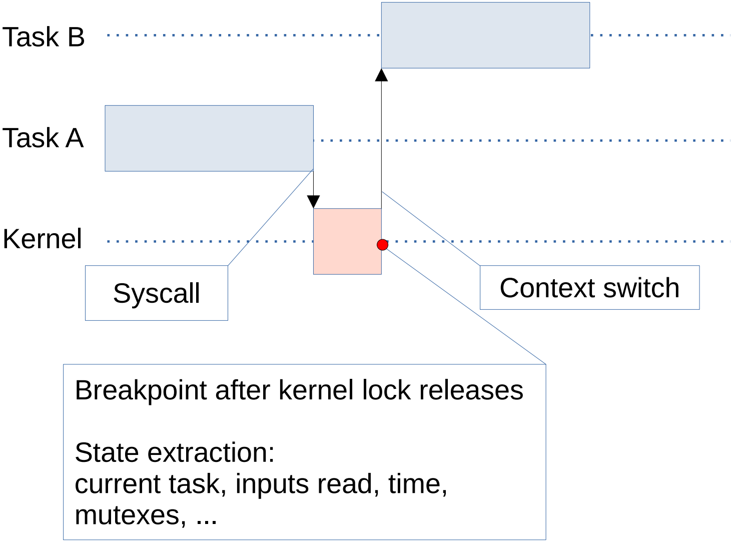
Emulator hosting RTOS

WCRT Fuzzers - Execution



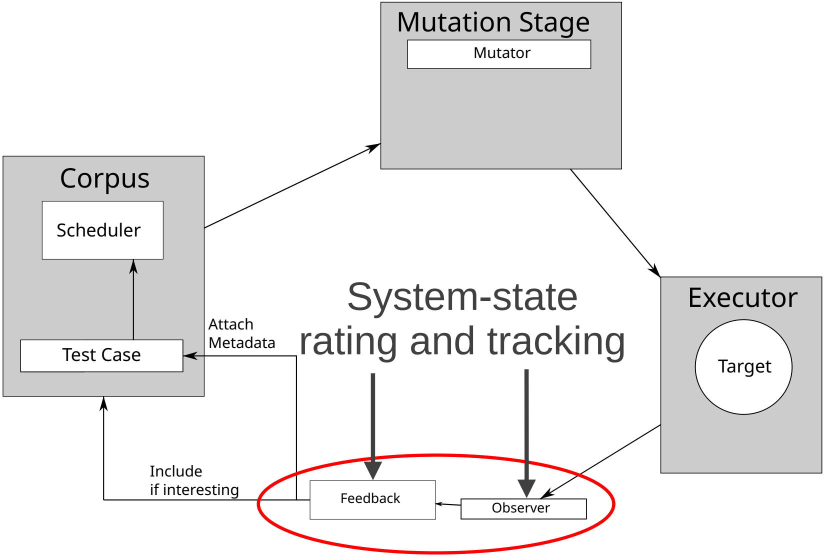
Emulation for tracing target OS data structures

WCRT Fuzzers - Execution


Task B
Task A
Kernel
Syscall
Context switch
Breakpoint after kernel lock releases
State extraction:
current task, inputs read, time, mutexes, ...

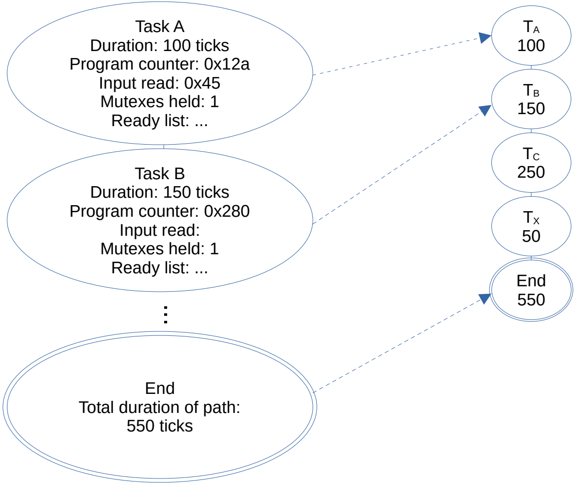
Extract state information at syscalls

WCRT Fuzzers - Rating



Feedback: Reward reaching new code or prolonging blocks of code

WCRT Fuzzers - Rating



Task A
Duration: 100 ticks
Program counter: 0x12a
Input read: 0x45
Mutexes held: 1
Ready list: ...

Task B
Duration: 150 ticks
Program counter: 0x280
Input read:
Mutexes held: 1
Ready list: ...

⋮

End
Total duration of path:
550 ticks

T_A
100

T_B
150

T_C
250

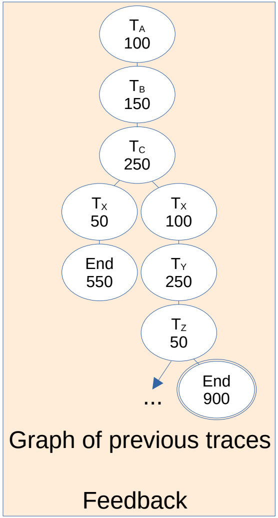
T_X
50

End
550

Trace of observed states

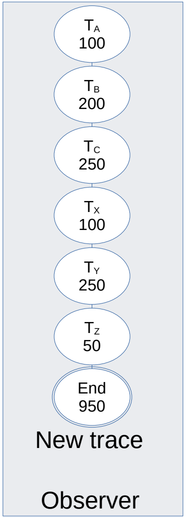
Simplified

WCRT Fuzzers - Rating



Graph of previous traces

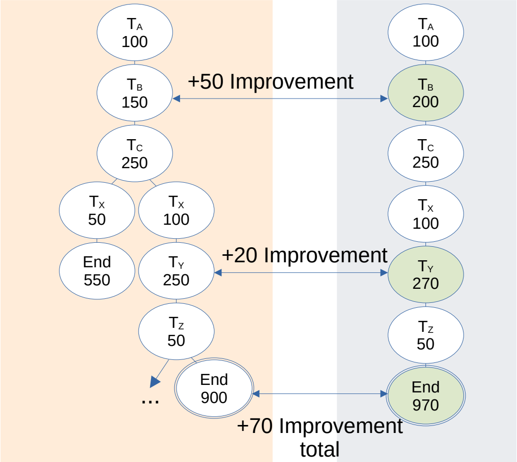
Feedback



New trace

Observer

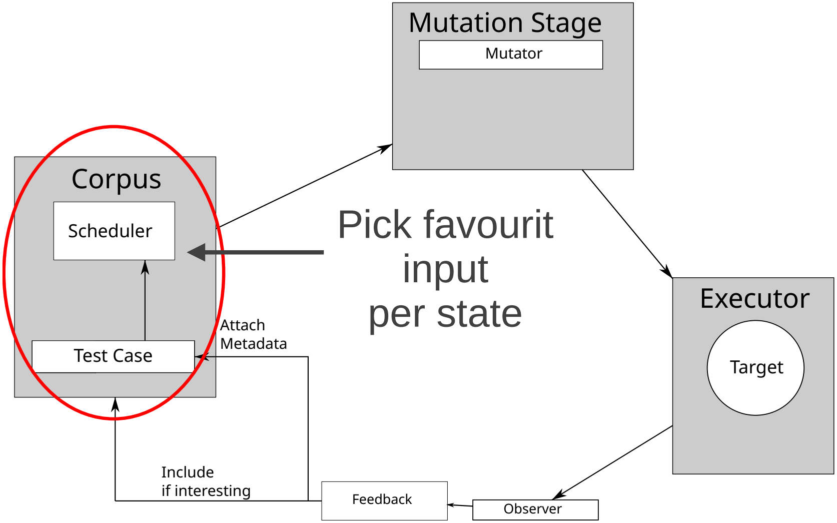
WCRT Fuzzers - Rating



Reward inputs for increasing execution time (multiple options)

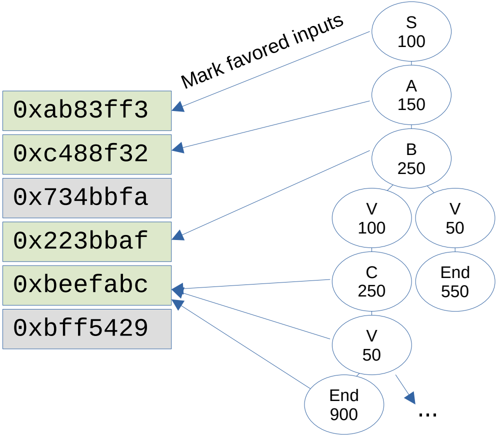
- Reward increases in single states
- Reward based on full state trace

WCRT Fuzzers - Prioritization



Scheduler: Favor the highest execution count/time per edge/state

WCRT Fuzzers - Prioritization

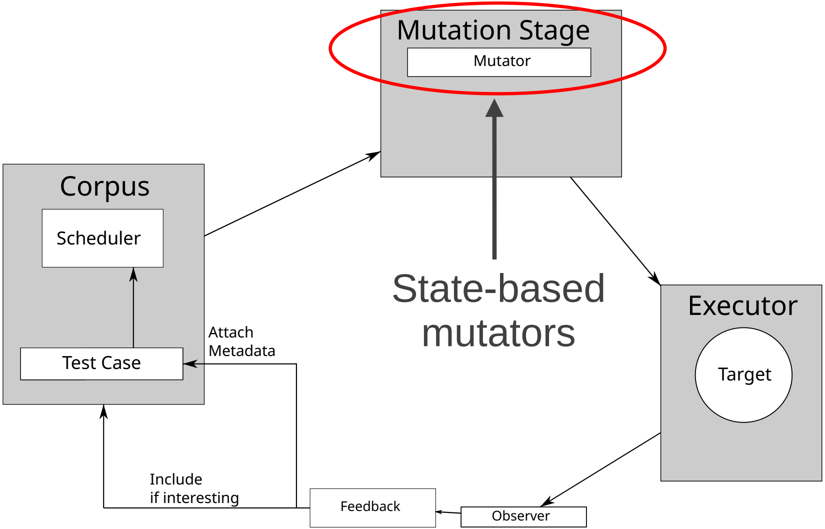


Inputs in corpus

State graph with metada

Scheduler: Favor the highest execution count/time per edge/state

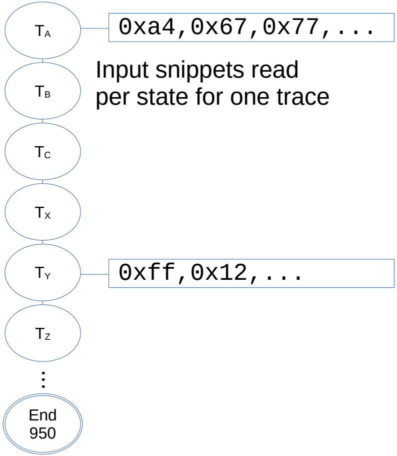
WCRT Fuzzers - Mutation



Mutator: Aim to create new states or maximize from past runs

WCRT Fuzzers - Mutation

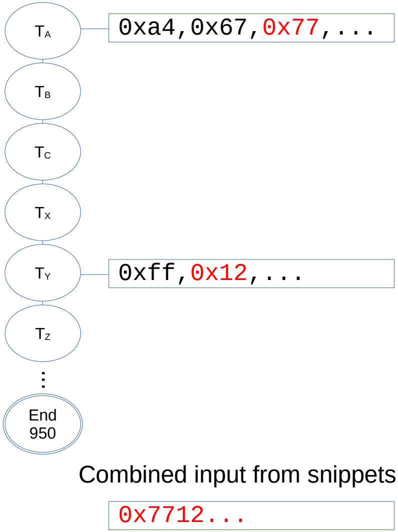
Crossover input snippets



Input snippets read per state for one trace

Aim to improve a trace by combining past input snippets of the same trace

WCRT Fuzzers - Mutation



T_A

0xa4,0x67,0x77,...

T_B

T_C

T_X

T_Y

0xff,0x12,...

T_Z

⋮

End
950

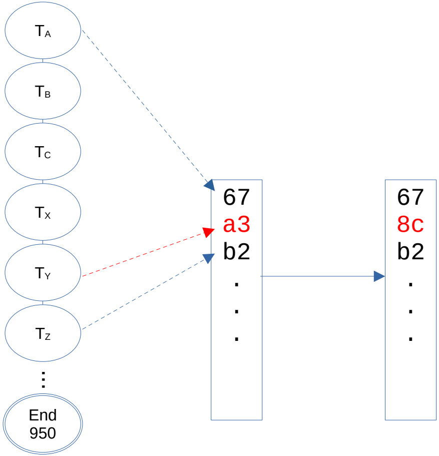
Combined input from snippets

0x7712...

Aim to improve a trace by combining past input snippets of the same trace

WCRT Fuzzers - Mutation

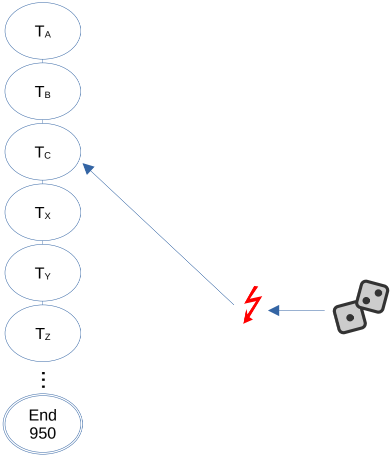
Randomize snippets



Aim to discover new states by changing snippets of an input

WCRT Fuzzers - Mutation

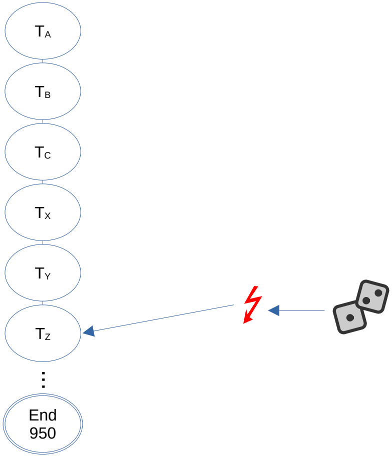
Alter interrupt timings



Aim to trigger new state-branches by placing the interrupt in different states

WCRT Fuzzers - Mutation

Alter interrupt timings



Aim to trigger new state-branches by placing the interrupt in different states

Outline

Introduction

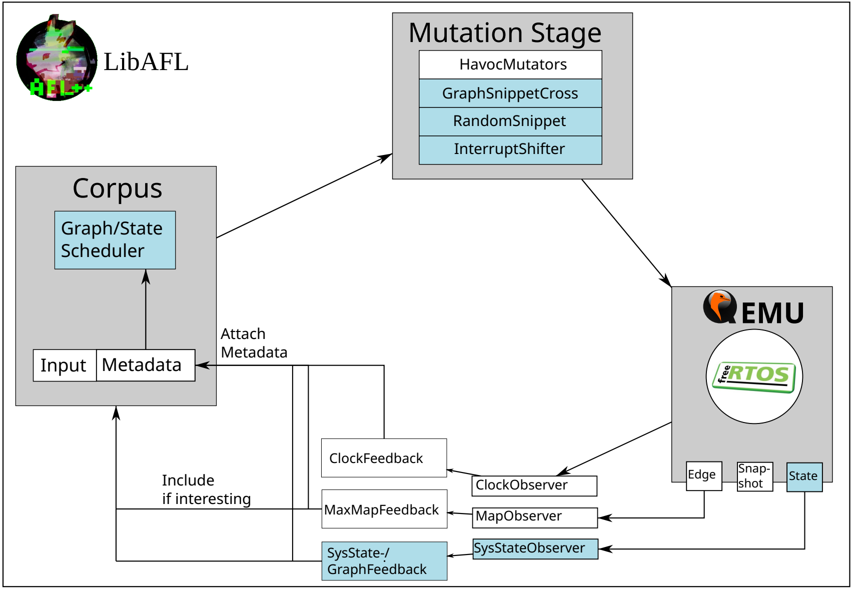
Approach: **F**uzzing **RE**sponse **T**imes (FRET)

FRET Implementation

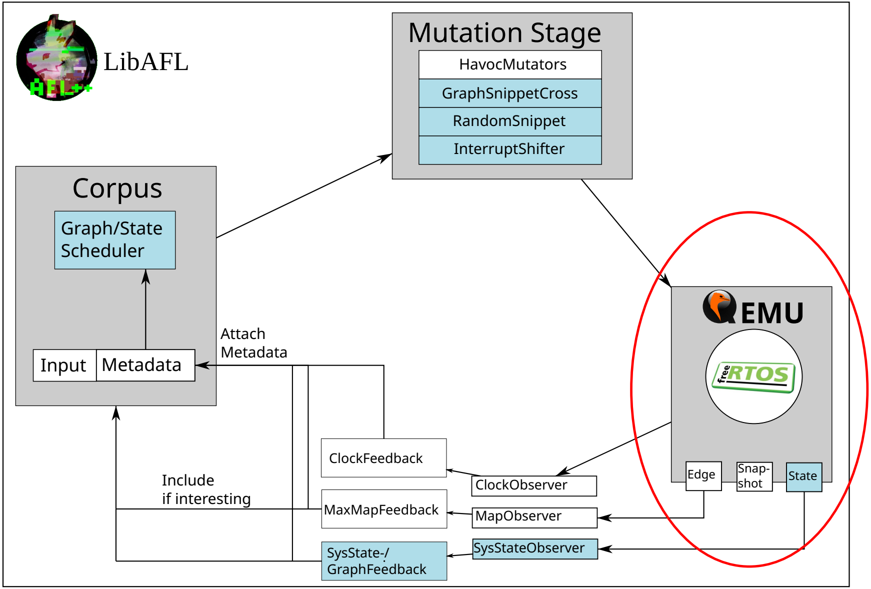
Evaluation

Summary

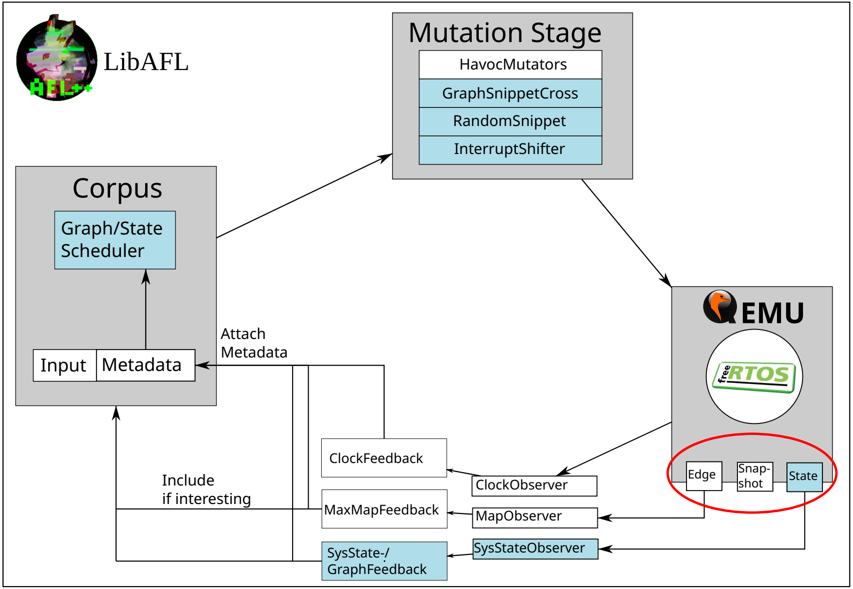
Prototype Implementation



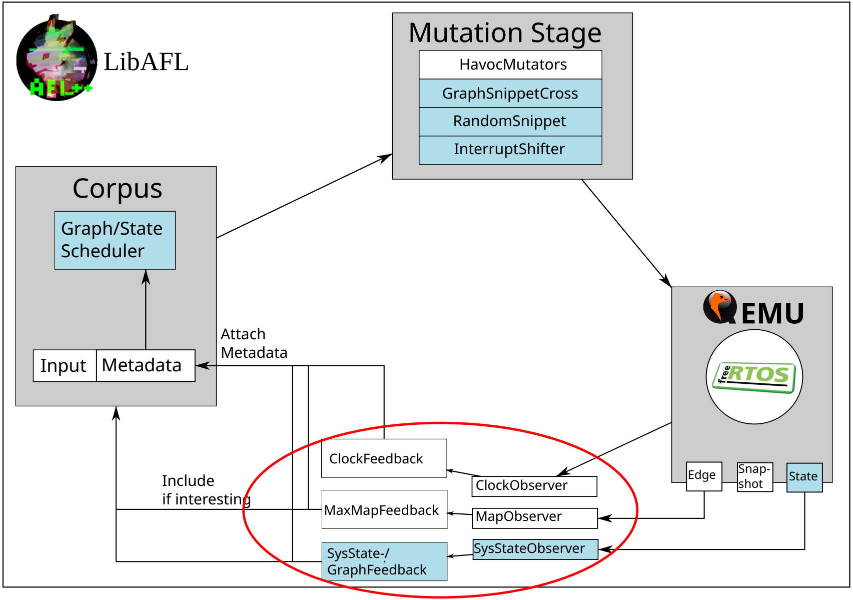
Prototype Implementation



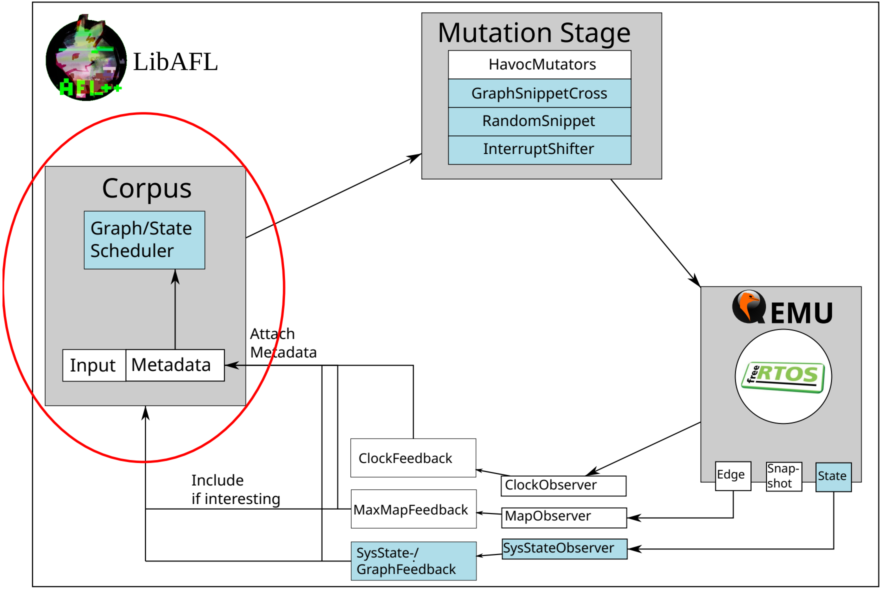
Prototype Implementation



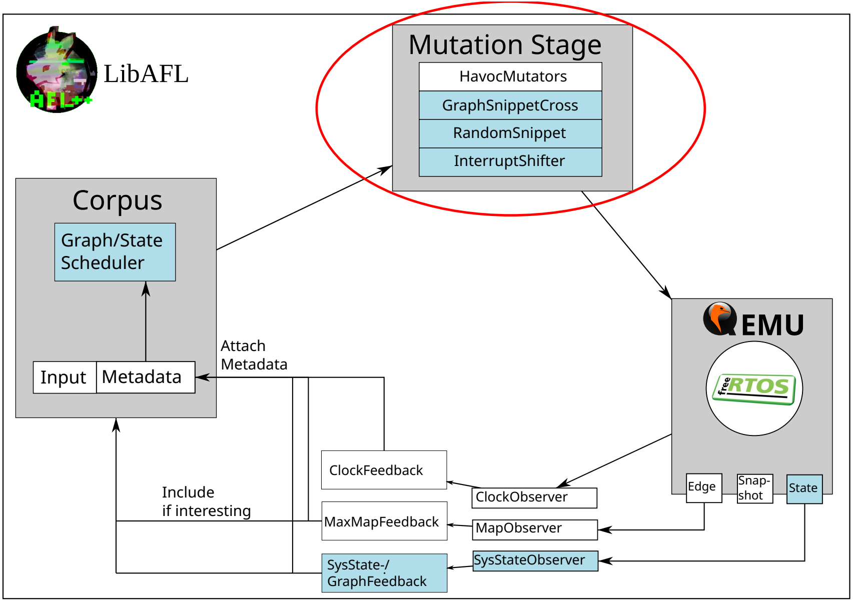
Prototype Implementation



Prototype Implementation



Prototype Implementation



Outline

Introduction

Approach: **F**uzzing **RE**sponse **T**imes (FRET)

FRET Implementation

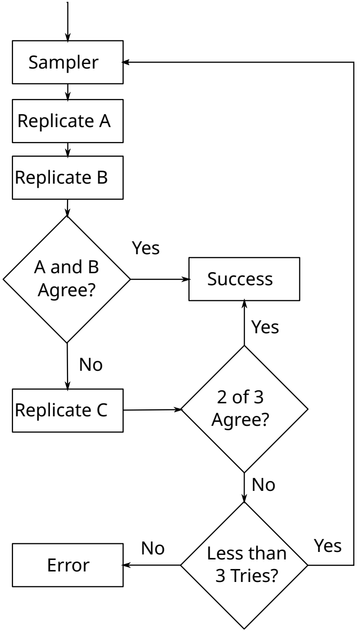
Evaluation

Summary

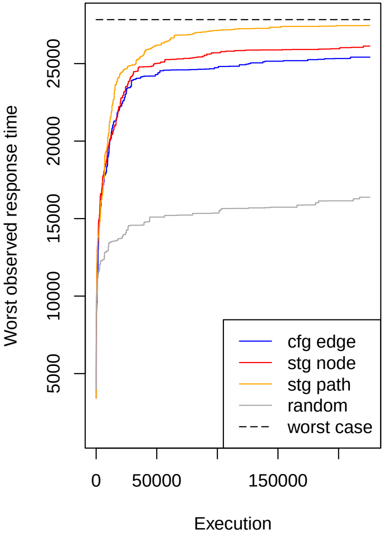
Evaluation Setup

Choose example with global and local optimization potential

- Aim to trigger maximum number of retries
- Maximize time per replicate

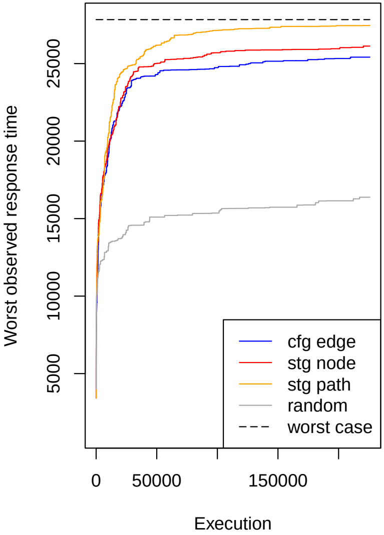


Preliminary Results

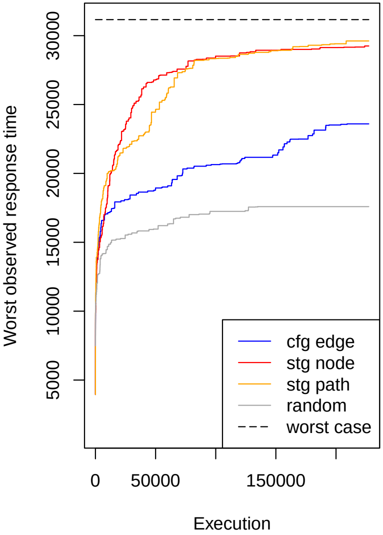


Cooperative scheduling

Preliminary Results



Cooperative scheduling



Preemptive scheduling with asynchronous events

Observed Corpus Sizes

Configuration	avg. corpus	σ_{corpus}
cfg edge	8,1	0,88
stg path	94,2	18,20
stg node	79,1	4,68

Cooperative scheduling

Configuration	avg. corpus	σ_{corpus}
cfg edge	340.3	2.31
stg path	1365.3	70.65
stg node	2392.3	111.46

Preemptive scheduling

→ CFG based fuzzing does not recognize all global paths as interesting

Observed Corpus Sizes

Configuration	avg. corpus	σ_{corpus}
cfg edge	8,1	0,88
stg path	94,2	18,20
stg node	79,1	4,68

Cooperative scheduling

Configuration	avg. corpus	σ_{corpus}
cfg edge	340.3	2.31
stg path	1365.3	70.65
stg node	2392.3	111.46

Preemptive scheduling

→ CFG based fuzzing does not recognize all global paths as interesting

→ Asynchronous events massively increase the state space

Observed Corpus Sizes

Configuration	avg. corpus	σ_{corpus}
cfg edge	8,1	0,88
stg path	94,2	18,20
stg node	79,1	4,68

Cooperative scheduling

Configuration	avg. corpus	σ_{corpus}
cfg edge	340.3	2.31
stg path	1365.3	70.65
stg node	2392.3	111.46

Preemptive scheduling

→ CFG based fuzzing does not recognize all global paths as interesting

→ Asynchronous events massively increase the state space

→ CFG based fuzzers are slowed down by edges produced by asynchronous events

Outline

Introduction

Approach: **F**uzzing **RE**sponse **T**imes (FRET)

FRET Implementation

Evaluation

Summary

Fuzzing REsponse Times (FRET)

- Whole system influences WCRT
- Mutation based fuzzing to maximize response time
- System-state based fuzzing

Method	Random	Fuzzing	FRET
Guided Exploration	✗	✓	✓
Time Maximization	✗	✓	✓
Reponse Maximization	✗	✗	✓

Future Work

- Cycle accurate emulation
- Eliminate redundant states