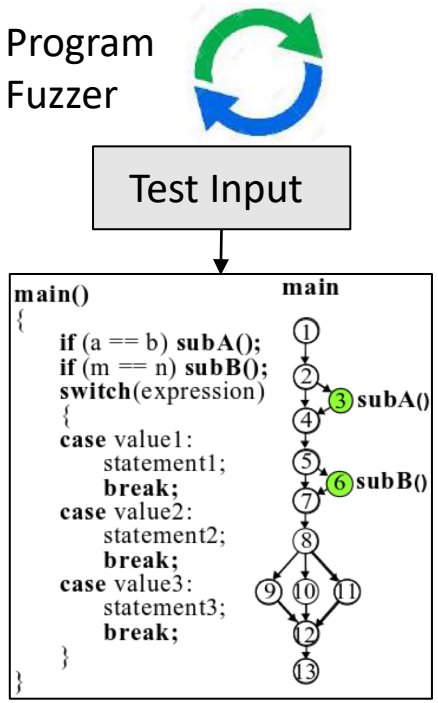
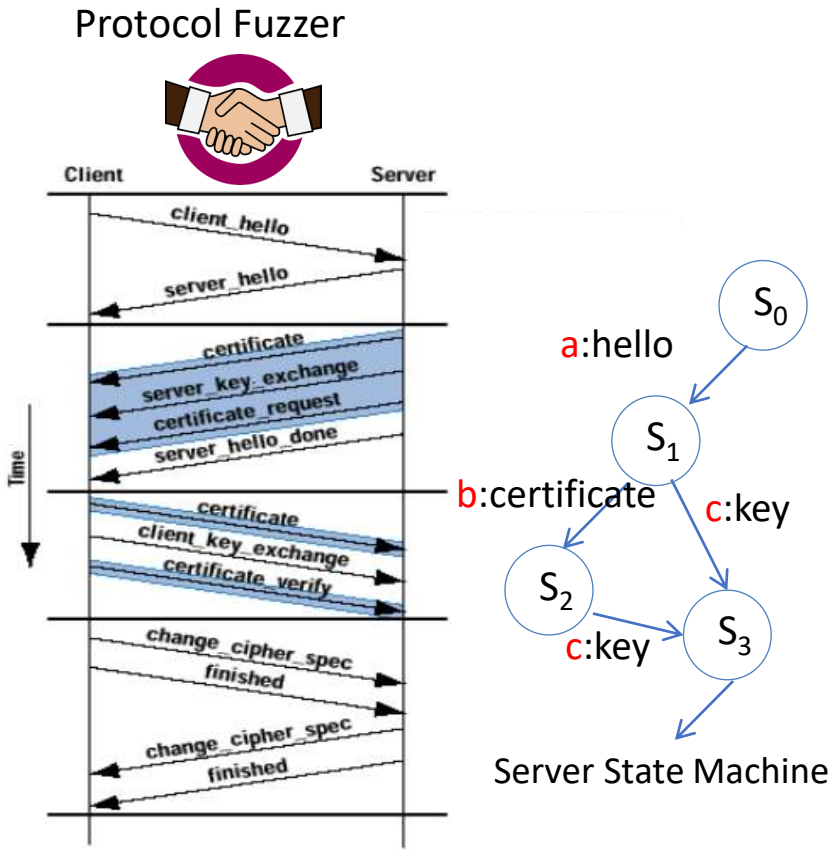
yFuzz: a yield-driven progressive fuzzer for stateful communication protocols

Yurong Chen, Tian Lan, Guru Venkataramani

Fuzzing communication protocols

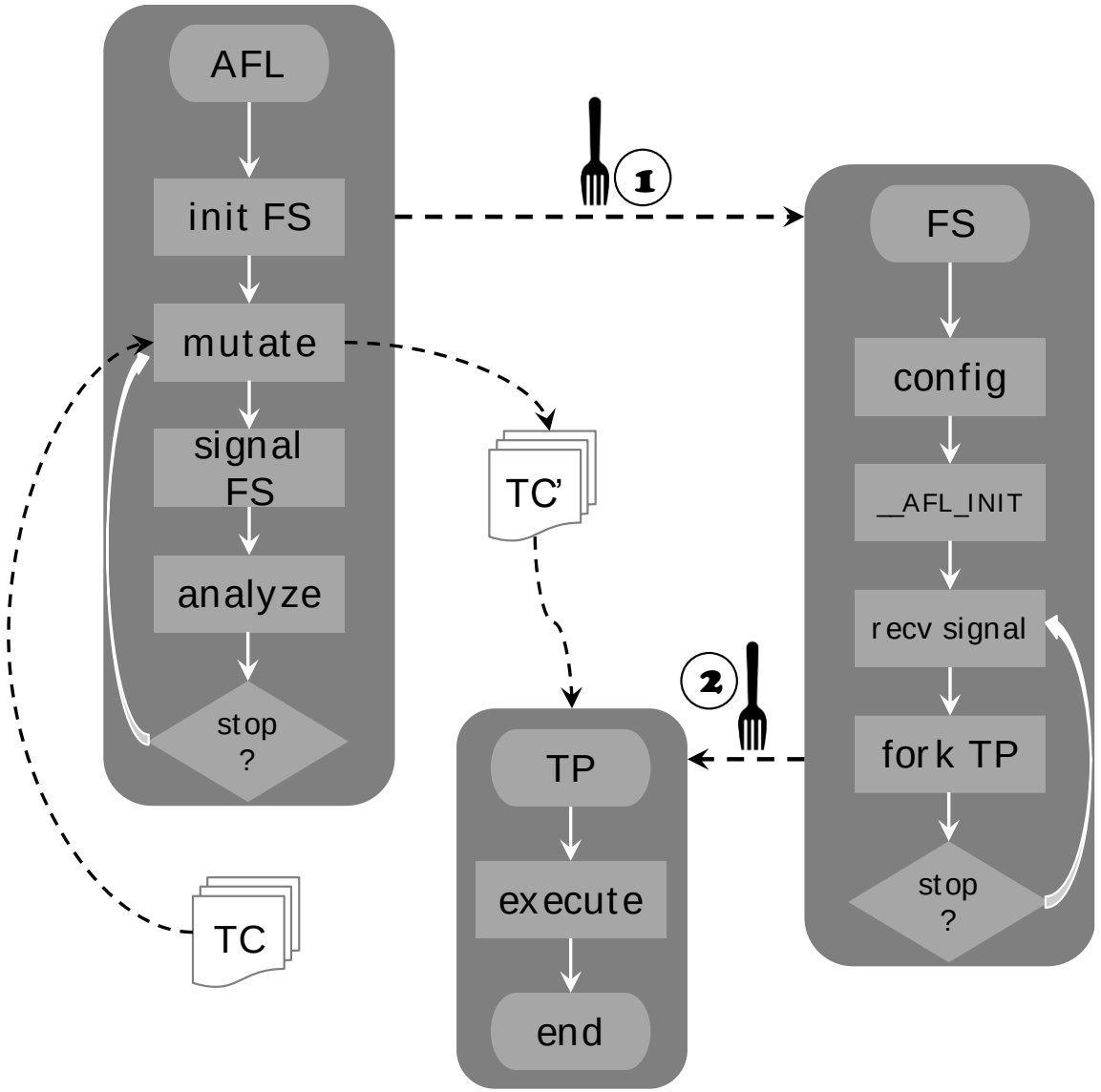




- Protocol fuzzing faces some very unique challenges:
  - Most communication protocols are stateful, so fuzzing should be state-aware
  - They involve multiple entities communicating with each other
  - Many packets/fields are interdependent and thus cannot be fuzzed separately

AFL Forkserver

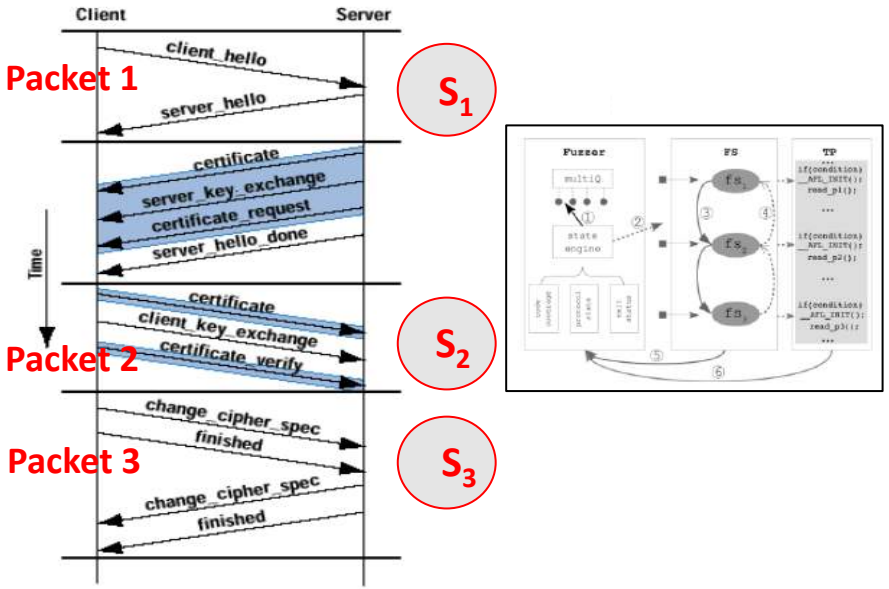
- Developed to avoid repeated "*execv()*", which is an expensive syscall. Instead, use *"fork()"* to spawn new fuzzing instances



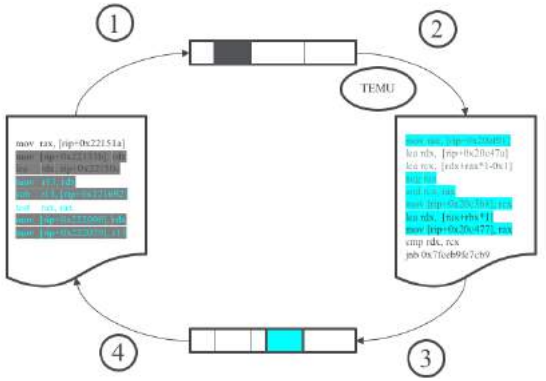
Our Solution: Progressive Protocol Fuzzing

- We propose a “dynamic state forking and replay” mechanism to
  - build multiple fuzzing states across the test program execution and switch between them.
  - replay any critical point of execution and mutate relevant packets, while maintaining protocol state consistency.
  - enable various fuzzing strategies such as “onion peeling”, “drill through”, and “greedy”.

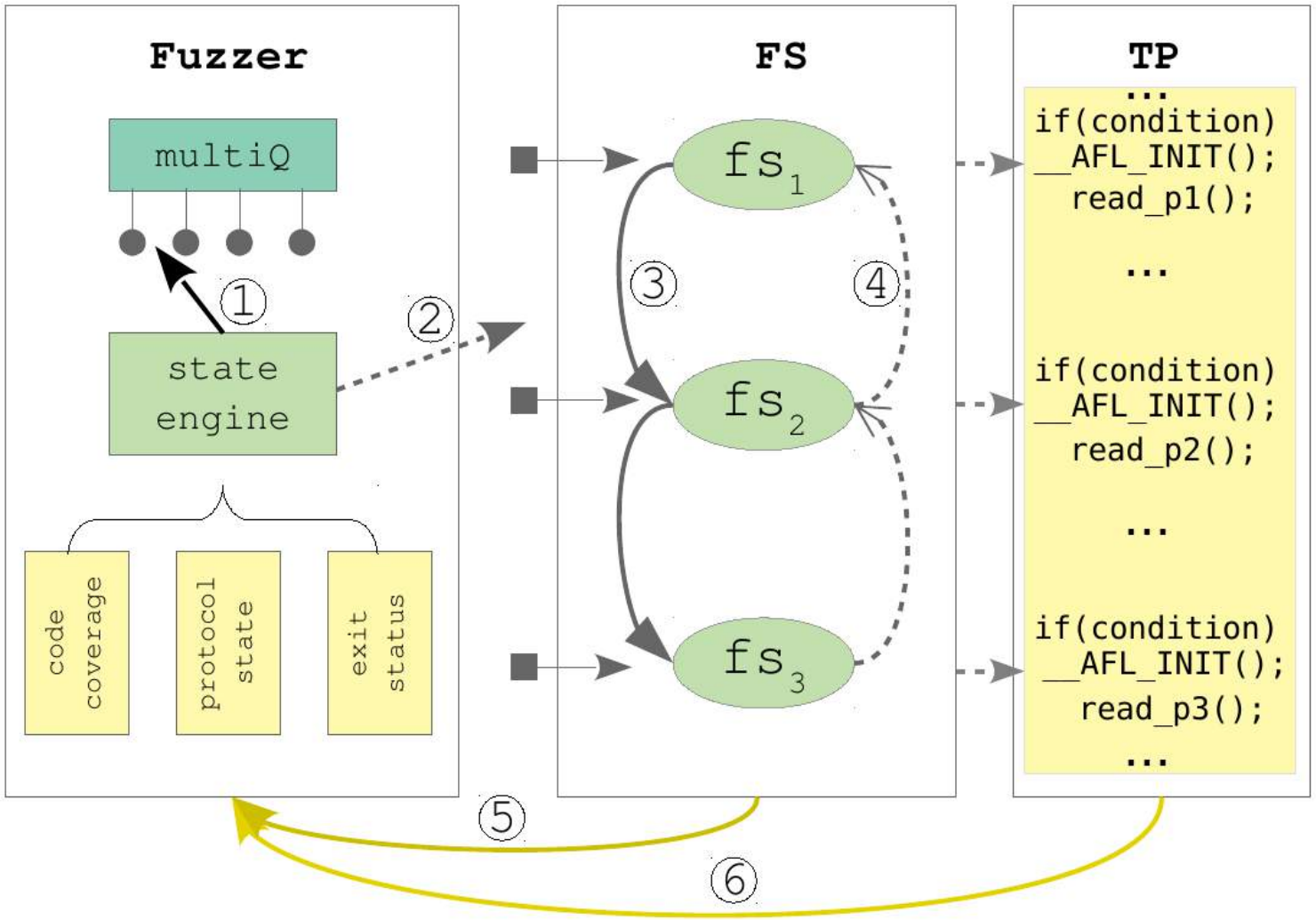
State forking & replay



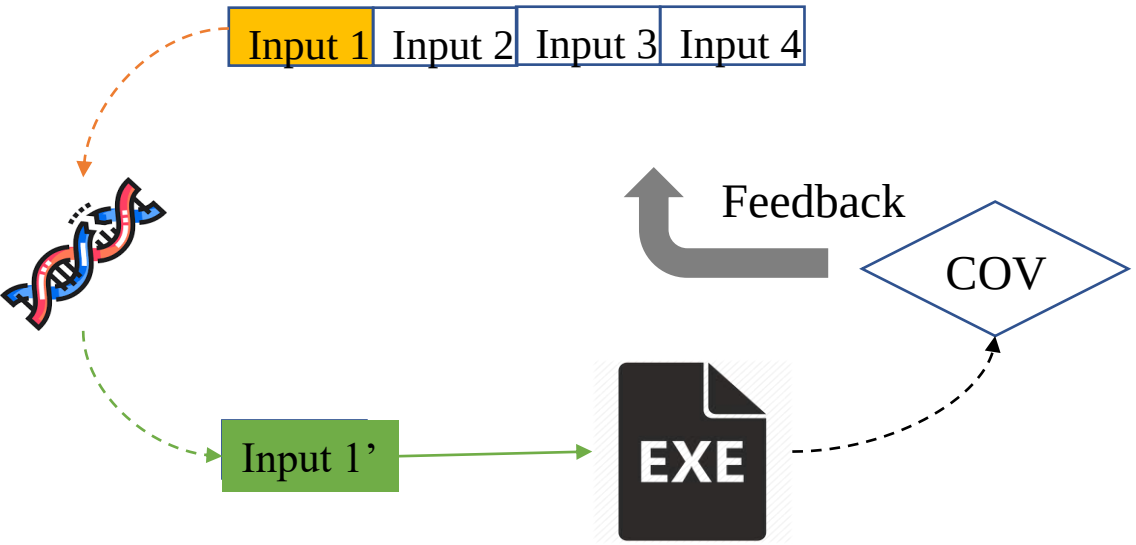
Cross-host tainting



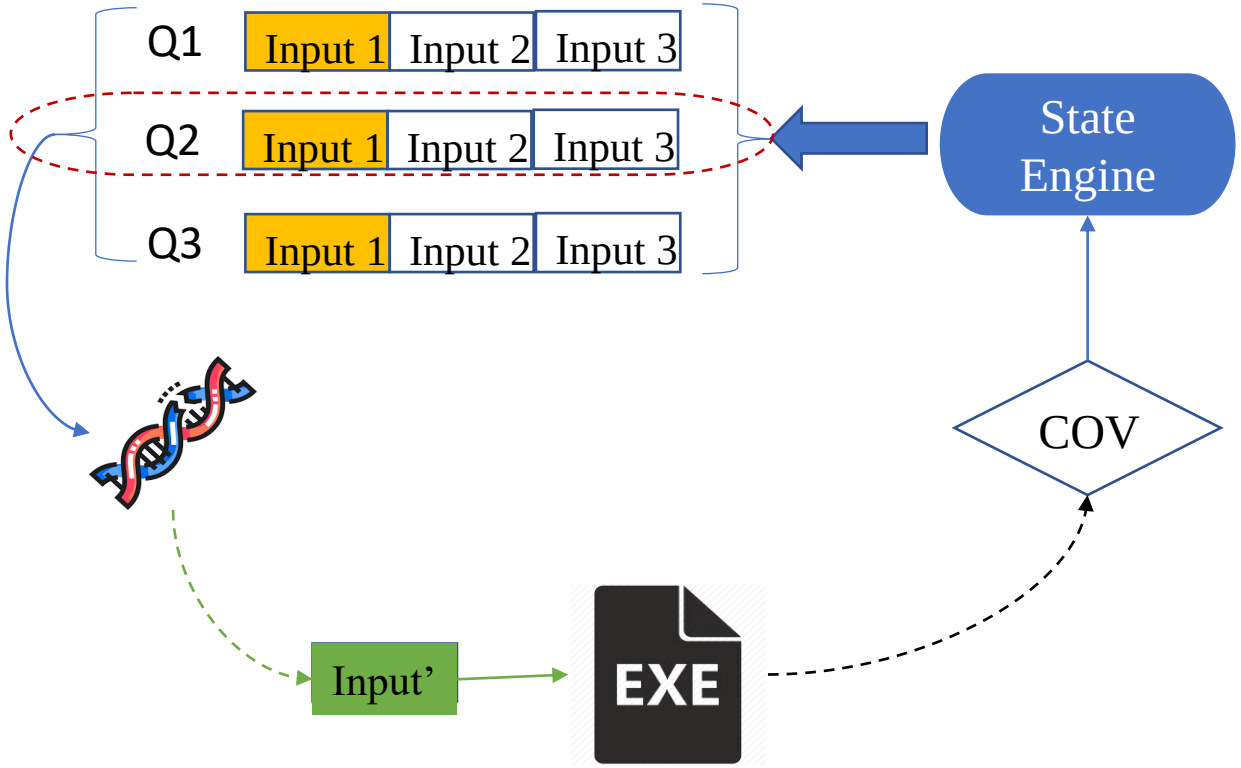
System design



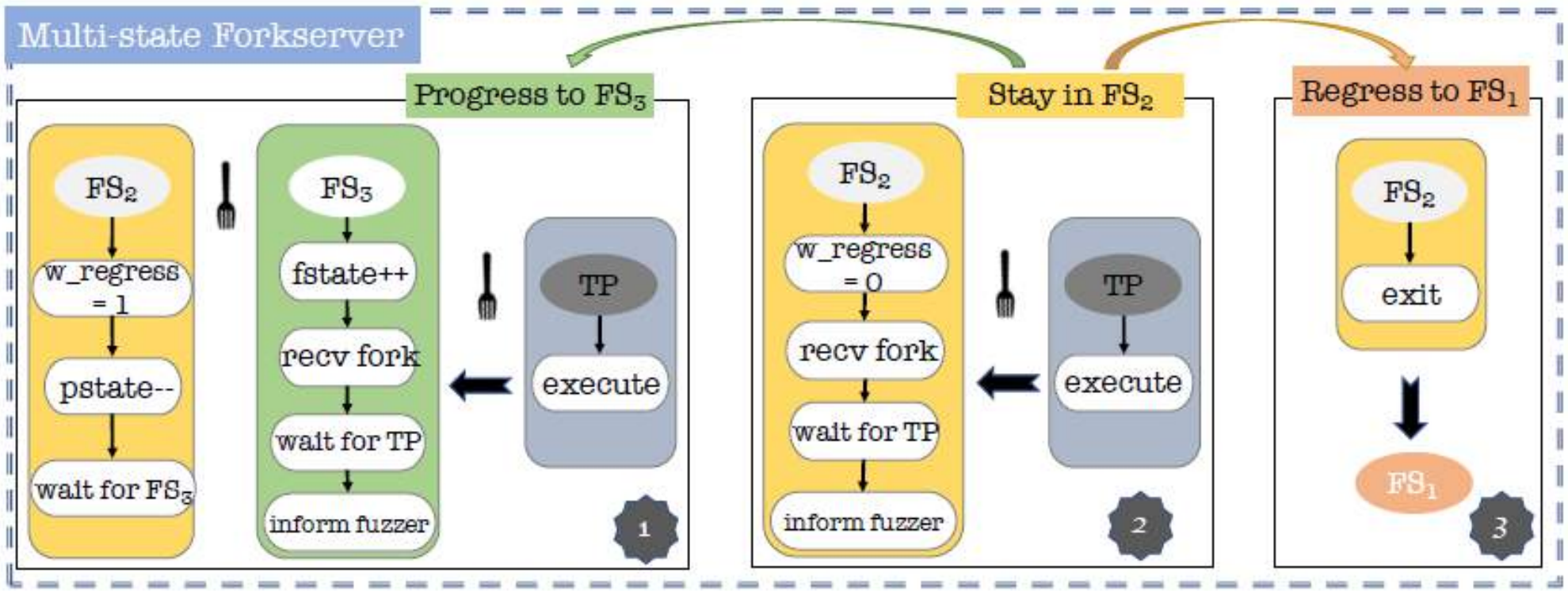
Non-stateful:



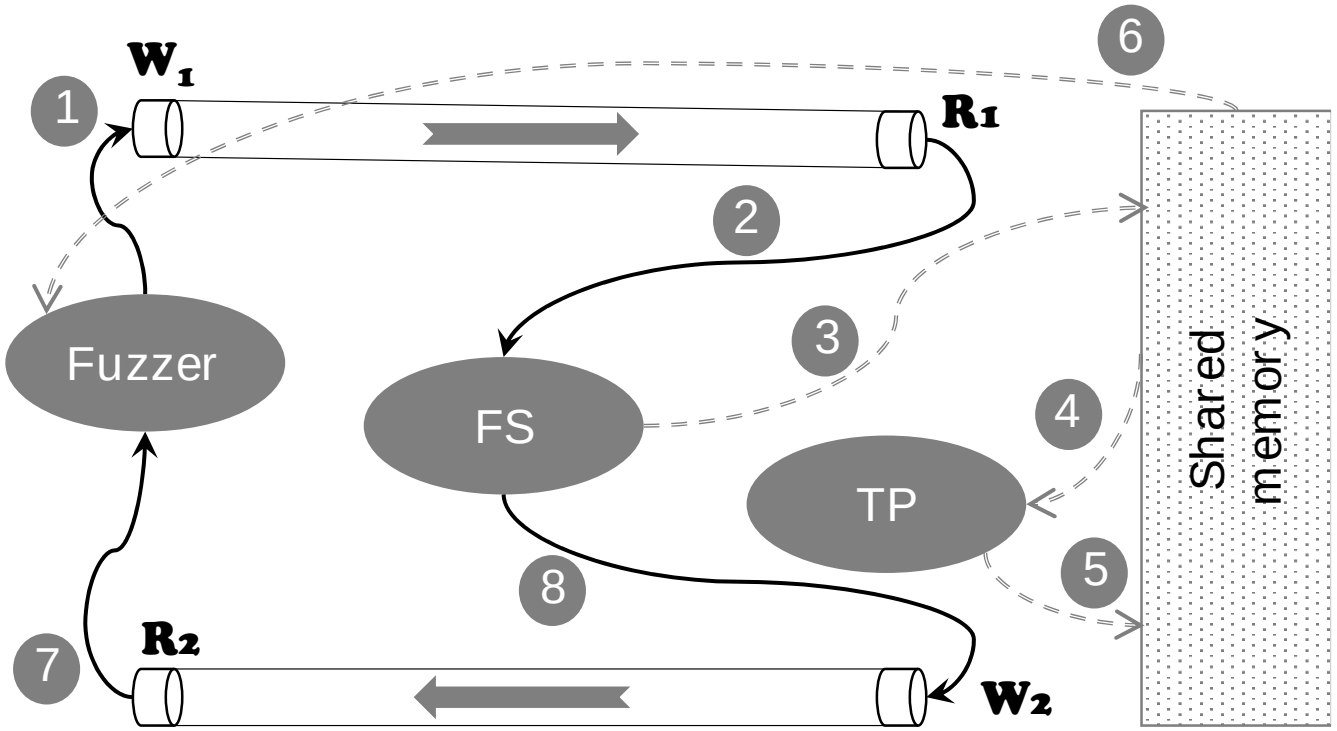
State-aware:



Stateful fuzzing engine

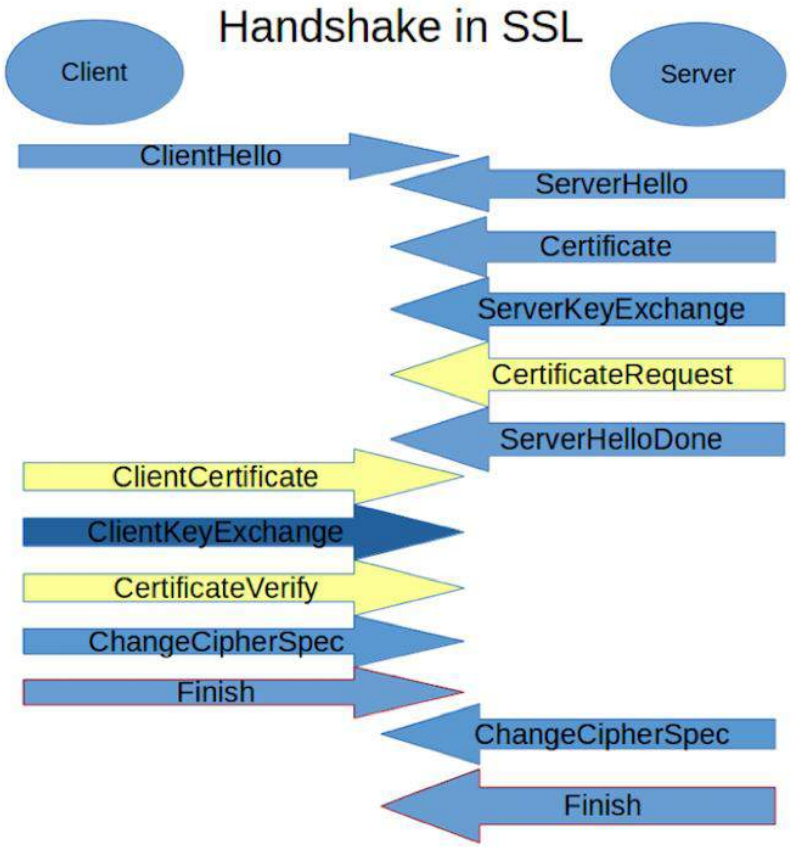


Communication among fuzzer, forkserver and testing program



1. Fuzzer signals FS to fork through pipe
2. FS receives signal
3. FS informs TP about fuzzing state
4. TP reads fuzzing state
5. TP writes fuzzing state
6. Fuzzer reads COV
7. Fuzzer reads TP ret status from pipe
8. FS writes TP ret status to pipe

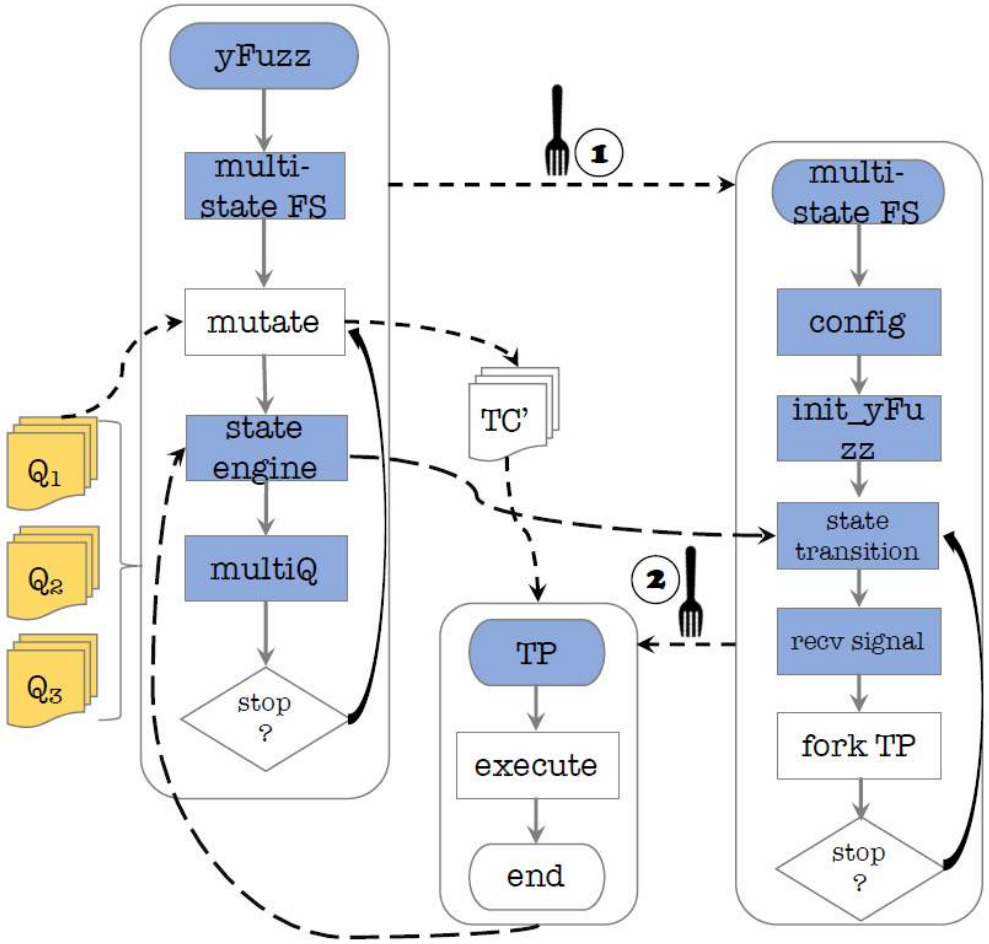
Evaluation on SSL Protocol



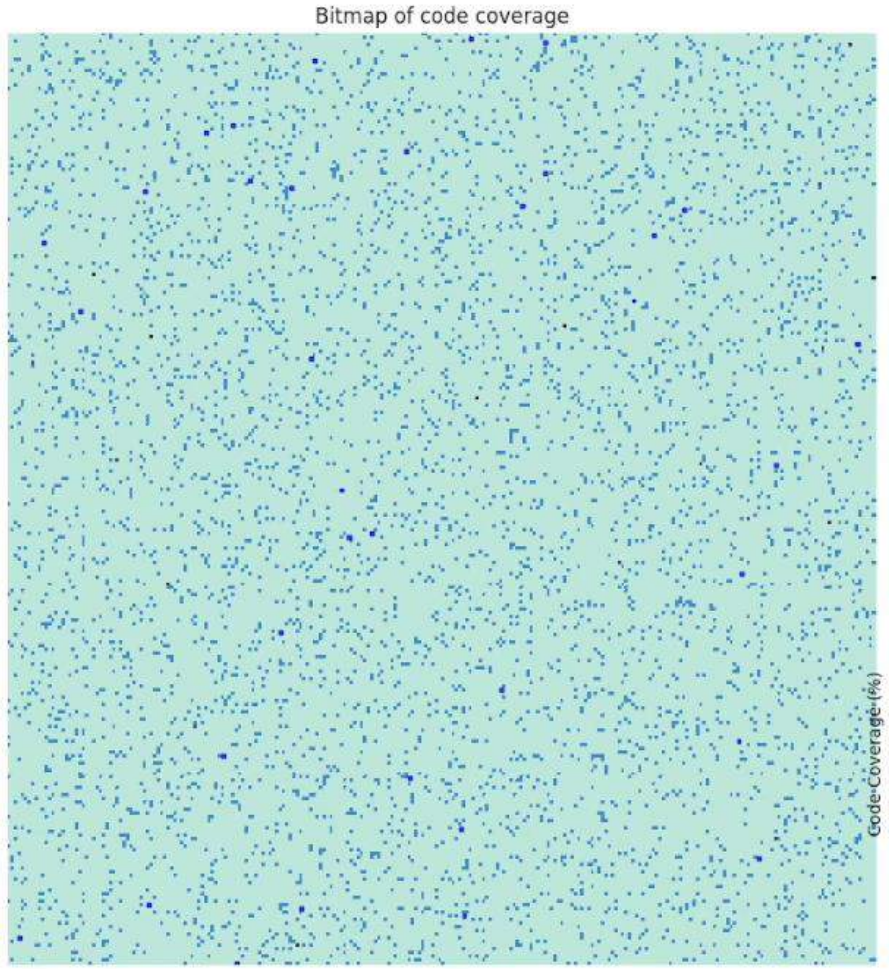
- A standard protocol for authentication and creating a secure connection.
  - Running in server/client mode.
  - Authentication through handshakes.
  - Also allowing other functionalities such as selecting cipher suites and deriving shared keys
  - Typically four flights in the handshake

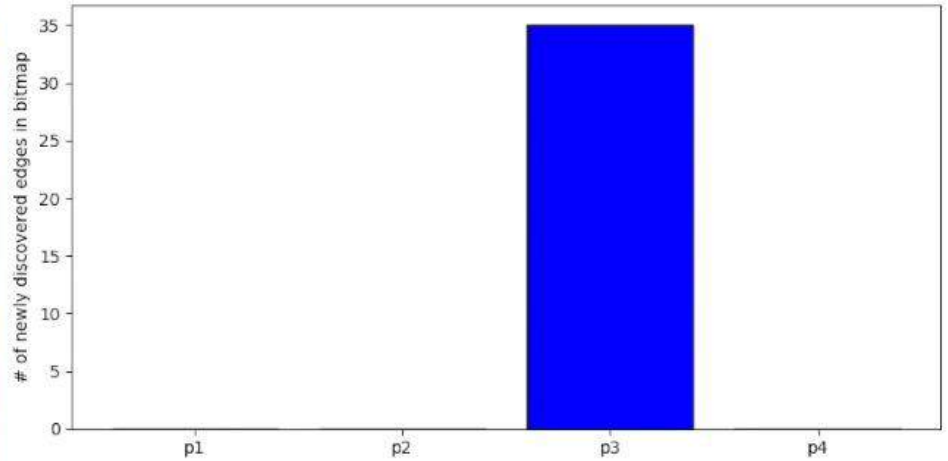
- Many vulnerabilities of OpenSSL are well documented in CVE and serve as ground truth.
  - Heartbleed (CVE-2014-0160)
  - 17% of the Internet's secure web servers certified by trusted authorities were vulnerable to the attack

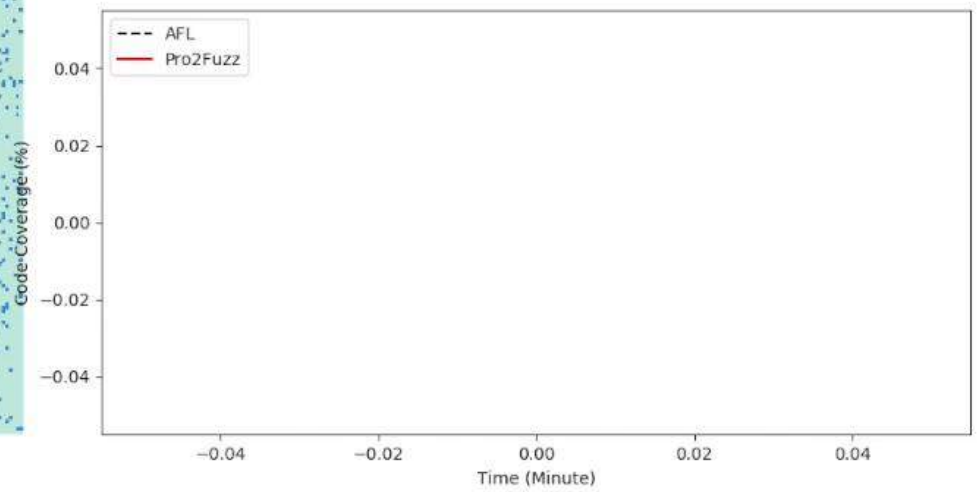
Implementation upon AFL



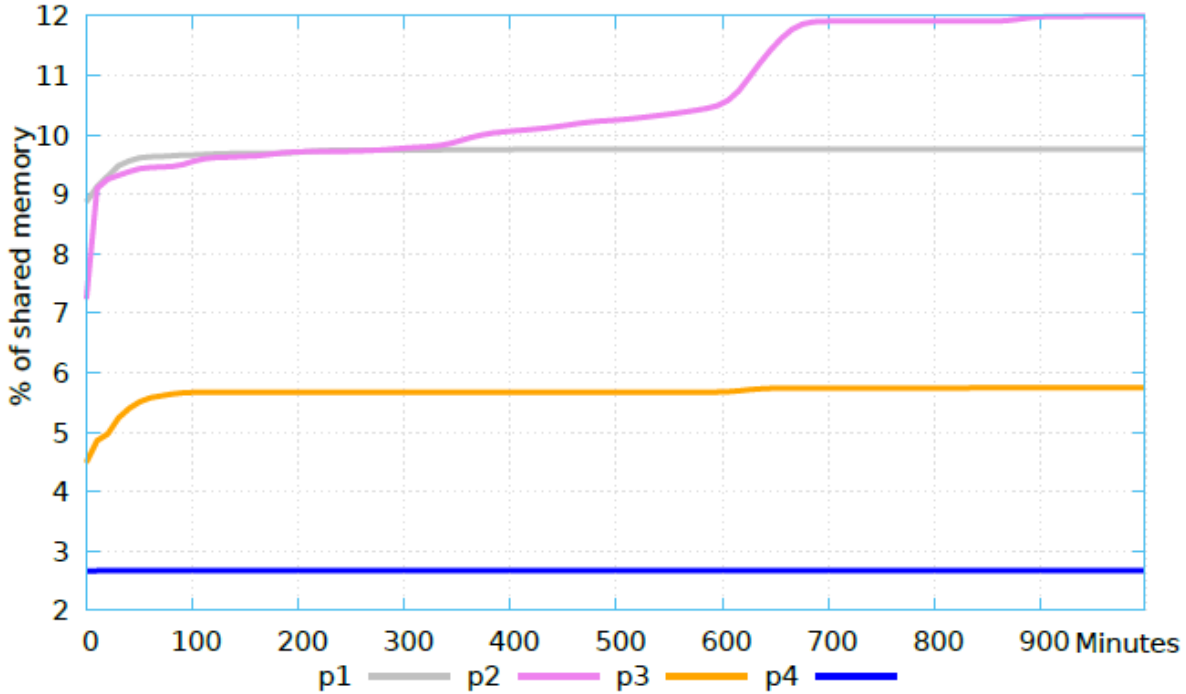
Demo

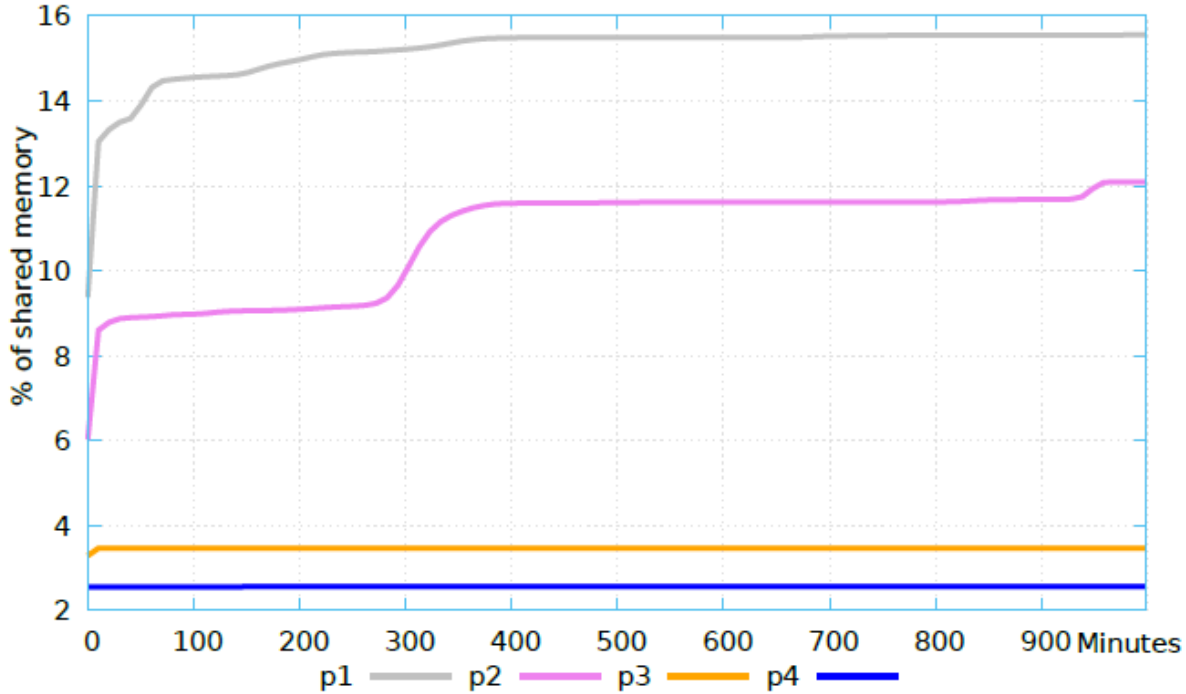

Bitmap of code coverage


# of newly discovered edges in bitmap
35
30
25
20
15
10
5
0
p1
p2
p3
p4


AFL
Pro2Fuzz
Code Coverage (%)
0.04
0.02
0.00
−0.02
−0.04
−0.04
−0.02
0.00
0.02
0.04
Time (Minute)

Code coverage breakdown





Code coverage trend of fuzzing single packet at four different stages with OpenSSL 101 and 110.

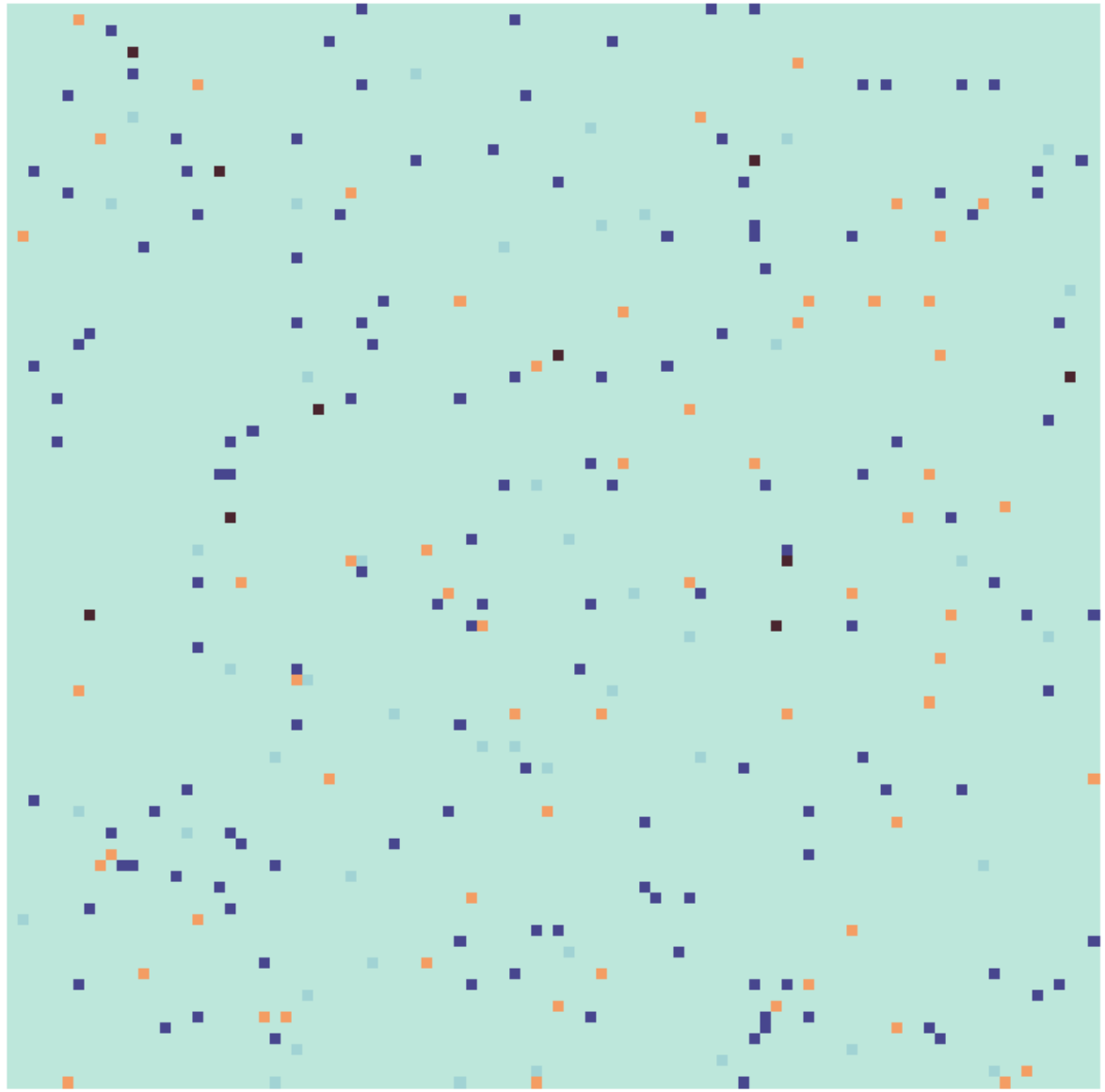


Fig. 9. Code coverage composition of fuzzing individual packet on OpenSSL v110. The 64kB shared memory is converted to a 256*256 map (only 100*100 is shown here because of the size limit), each cell representing an unique edge. Difference color represents execution edges that are discovered by fuzzing different packets.

Performance comparison

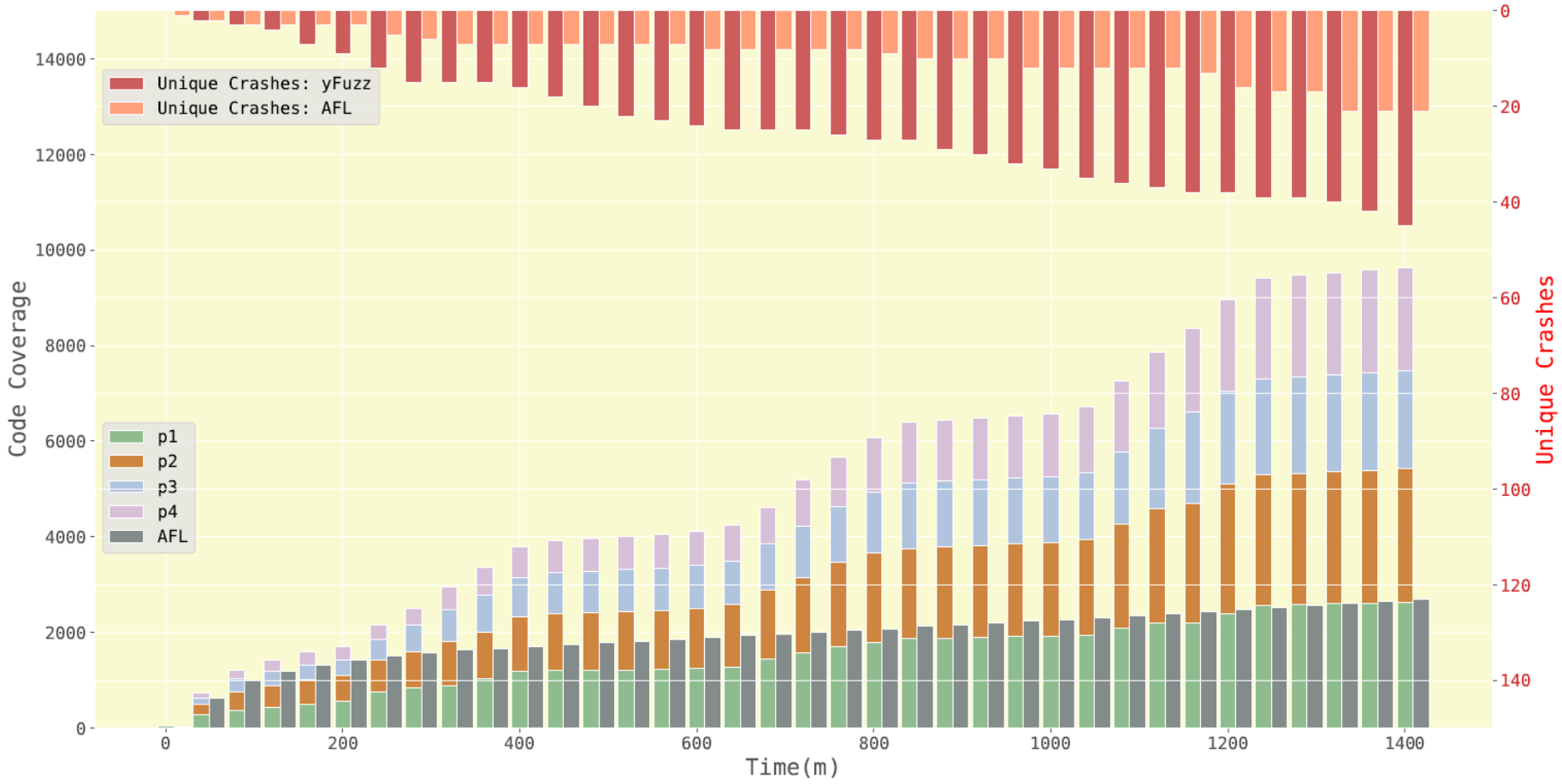


Fig. 11. AFL vs yFuzz : bottom bars show the composition of code coverage and top bars show the number of unique crashes. Note that we subtract the initial code coverage (7.9% in this case) explored by the seed $p1$ from each bottom bar to show the increments of code coverage. And the code coverage is represented as the number of discovered edges.

Table 1: Statistics of fuzzing single packet (OpenSSL v101) at four different stages using default AFL for 6 and 24 hours.

	Code Coverage(%)	Unique Crashes	Cycles Done	Total # of Executions(M)	Time (hours)
p1	9.51	1	4	7.87	6
p2	10.18	9	0	12.68	6
p3	5.56	9	15	12.21	6
p4	2.61	6	157	12.43	6

	Code Coverage(%)	Unique Crashes	Cycles Done	Total # of Executions(M)	Time (hours)
p1	9.64	11	30	42.05	24
p2	11.16	9	6	49.58	24
p3	5.6	14	410	66.20	24
p4	2.61	9	1308	54.80	24

Single packet fuzzing

OpenSSL

Table 2: Code coverage breakdown: the code explored by fuzzing four individual packet. Time is in hours. The total size of bitmap is 64kB (65536 Bytes). U*i* stands for the number of edges that are only explored when fuzzing packet *i* but not explored when fuzzing other packets (i.e., edges that is unique to packet *i*).

Version	Time	Covered	Uncovered	U1	U2	U3	U4
101	6	7677	57859	563	955	30	386
	10	8879	56657	373	962	29	360
	24	8896	56640	312	966	32	359
110	6	10721	54815	1472	755	26	296
	10	10093	55443	2123	81	15	293
	24	11272	54264	2054	738	22	295

Stateful fuzzing

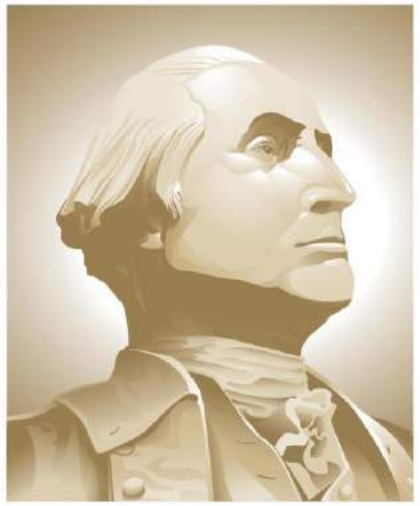
OpenSSL

Search Policy

- Progression
  - Current state f1, when interesting p1 triggers new code coverage, trap f1 and fuzz p2
- Regression
  - Current state f2, when fuzzing p2 is exhausted, step back to previous state f1, and continue fuzzing p1
- DFS backtracking
  - f1 => f2 => f3 => f4 => f3’ => f2’ => f1’

Search Policy: what to monitor?

- Key: variables that indicates state changes in the protocol
- Indicators:
  - Code coverage
  - New packet flight
  - Packet change: packet type, length, or combined signature



THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

N00014-17-1-2786 & N00014-15-1-2210

Thank you!