TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing

Yong-Hao Zou and Jia-Ju Bai, Tsinghua University; Jielong Zhou, Jianfeng Tan,

and Chenggang Qin, Ant Group; Shi-Min Hu, Tsinghua University

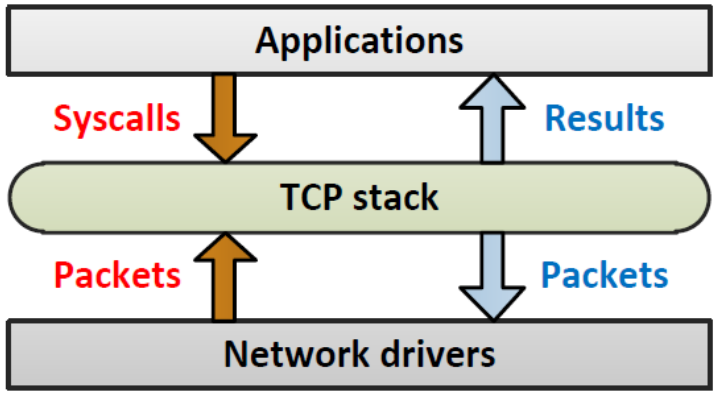
Background

➢ TCP Stack

- The transport layer protocol of the Internet(or **implementation**).

➢ Black box description

- Input
  - Syscall → `bind(),socket(),listen(),...`
  - Packets → TCP segment from IP layer
- Output
  - Syscall result → File descriptor from `socket()`,...
  - Packets → TCP segment to IP layer



Background

➢ Features of TCP Stack

- Dependency between the two inputs

  The input sequence of the TCP protocol stack has certain characteristics

  - **Syscall - syscall**

  `socket()`, `bind()`, `listen()` are often together and in a fixed order

  - **Packets - packets**

  The segments of the same TCP connection have the same IP address and port

  - **Syscall - packets**

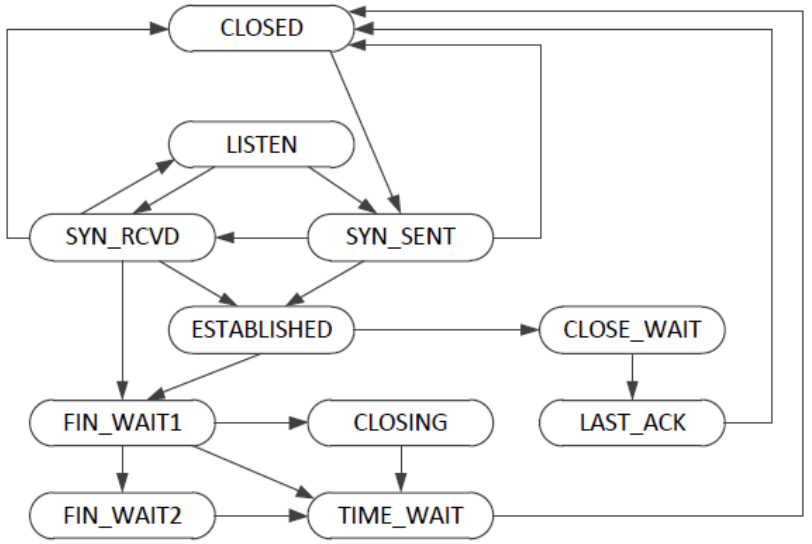
  `accept()` returns only after the TCP stack receives the last one of the three-way handshake packets.

Background

➢ Features of TCP Stack

- State model

  TCP Stack implementations have complex state models.

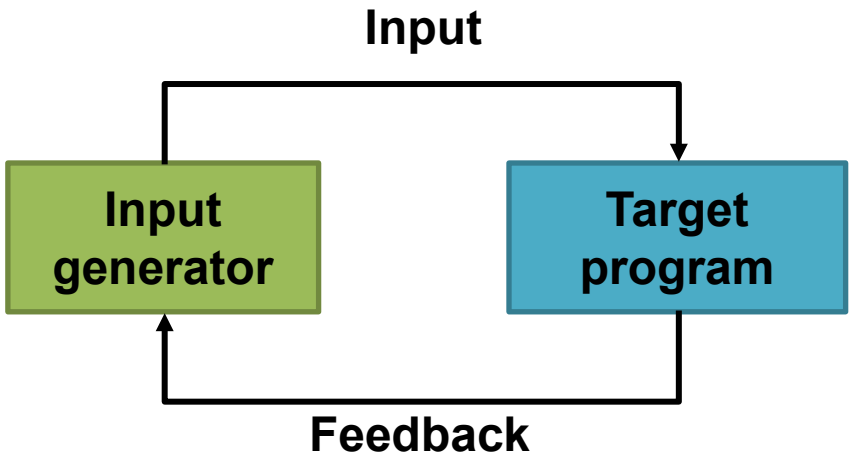


- Same Semantic rules

  The RFC document defines various behaviors of TCP.

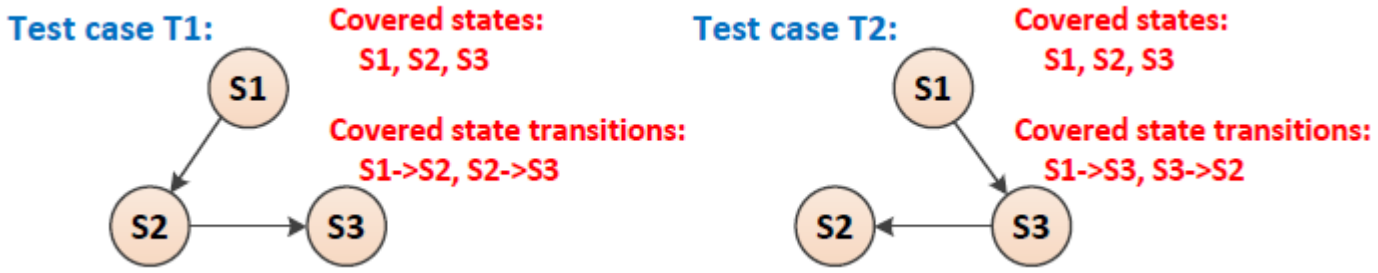
Background

- **Fuzz testing (Fuzzing)**
  - Automated software testing technique
  - Input(test case) generator
    - mutation-based(**need seeds**(input given by user))
    - generation-based(without seeds)
  - Application
    - File format
    - **Network protocol**



Problems

- Limitations of existing fuzzing for TCP Stack
  - Fail to generate two-dimensional inputs with dependencies;
  - Neglect the coverage of state transitions.



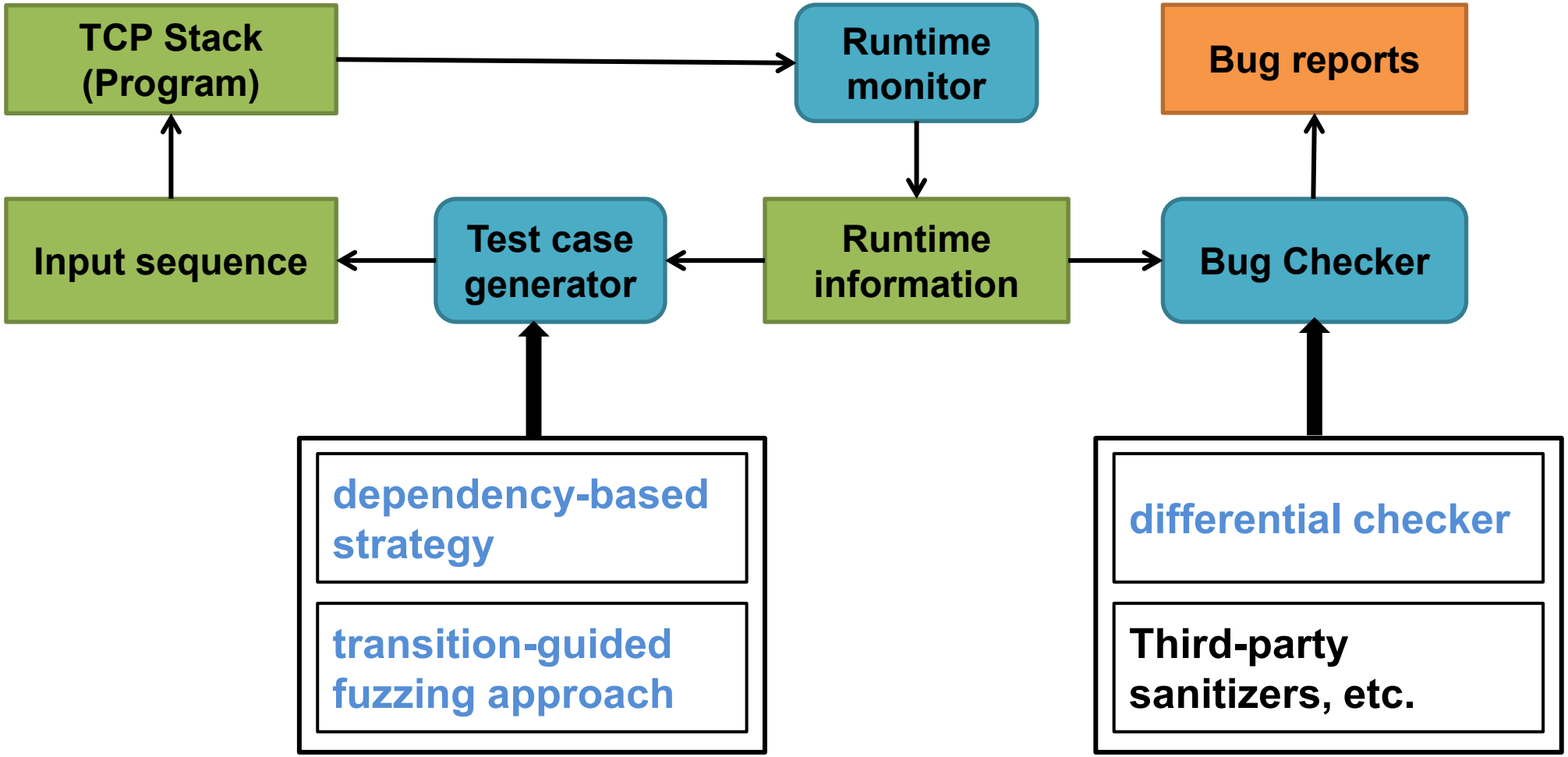
- Lack effective detection of semantic bugs

Time	FreeBSD		mTCP		TLDK	
	Memory	*Semantic*	*Memory*	*Semantic*	*Memory*	*Semantic*
2017	2	26	2	6	1	11
2018	9	51	0	4	0	4
2019	9	65	1	3	2	5
Total	20	142	3	13	3	20

Ideas

Features	Problems	Solution
Dependency between the two inputs	**Lack of dependency on input generated by the test**	**dependency-based strategy** to generate effective test case
State model	**Neglect the coverage of state transitions**	**transition-guided fuzzing approach** to improve the coverage of state transitions
Same Semantic rules	**Lack effective detection of semantic bugs**	a **differential checker** to detect semantic bugs

Design


TCP Stack (Program)
Runtime monitor
Bug reports
Input sequence
Test case generator
Runtime information
Bug Checker
dependency-based strategy
transition-guided fuzzing approach
differential checker
Third-party sanitizers, etc.

Dependency-Based Strategy

- Discard input sequences that do not meet certain dependency rules;
- Generate sequence according to dependency rules;
  - Dependency rules are defined in advance
- Rule Example:

Kind	**Dependency rule**
Syscall-syscall(x5)	`Socket()` and `connect()` are called in order when a connection is active open.
Packet-packet(x5)	After a connection is established, the source port and destination port of each packet are fixed
Syscall-packet(x5)	After close is called, a packet with the `FIN` flag should be sent

Transition-Guided Fuzzing Approach

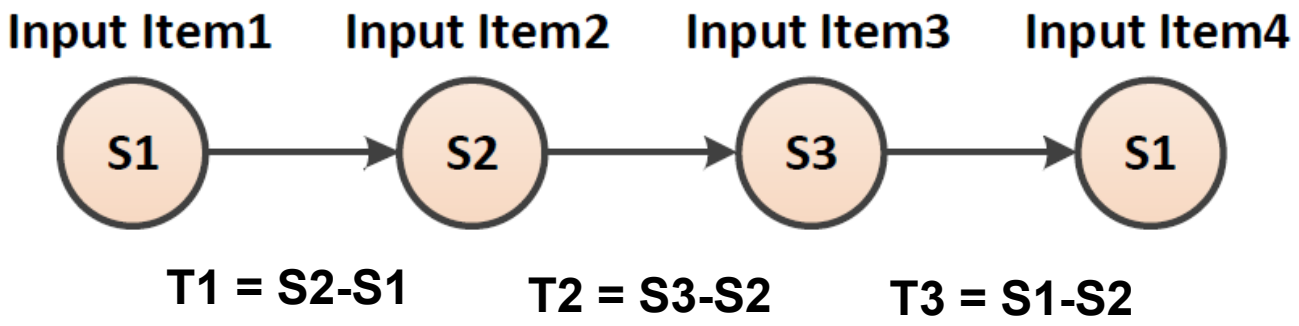
- State
  - Branch coverage (as existing fuzzing approaches do)
  - An example state `S1 = {1,0,1,0}`

BR1	BR2	BR3	BR4
1	0	1	0

- State transitions
  - State difference $BT = S_{i+1} - S_i$ (new)

Transition-Guided Fuzzing Approach

➢ Example

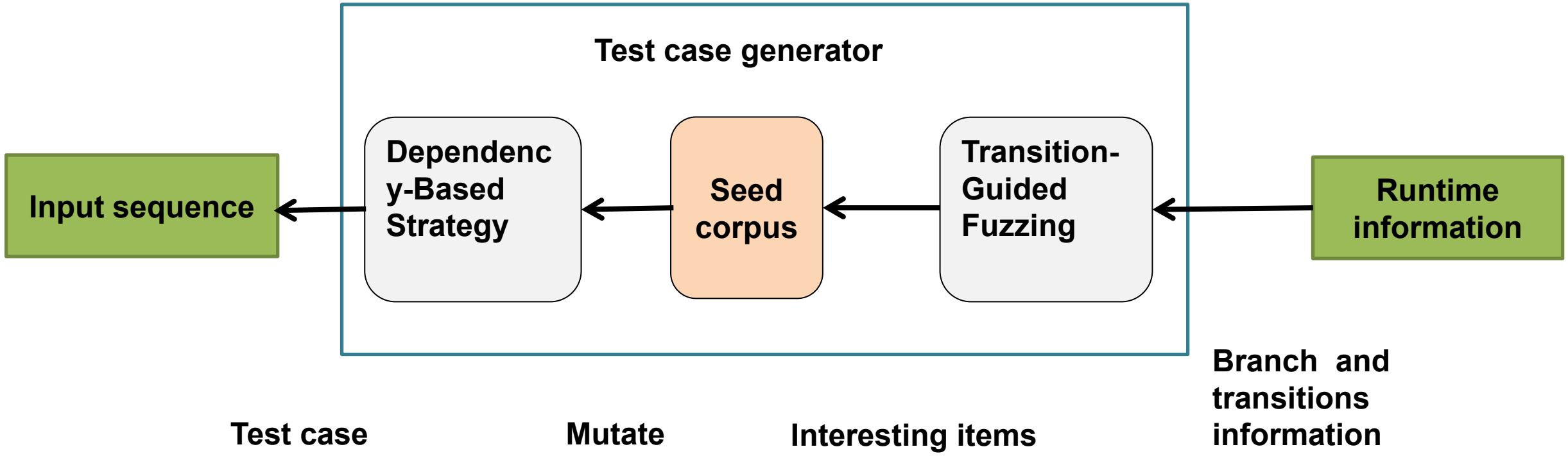


When input item4 :

State coverage only: S1 has appeared → input item4 is **useless**

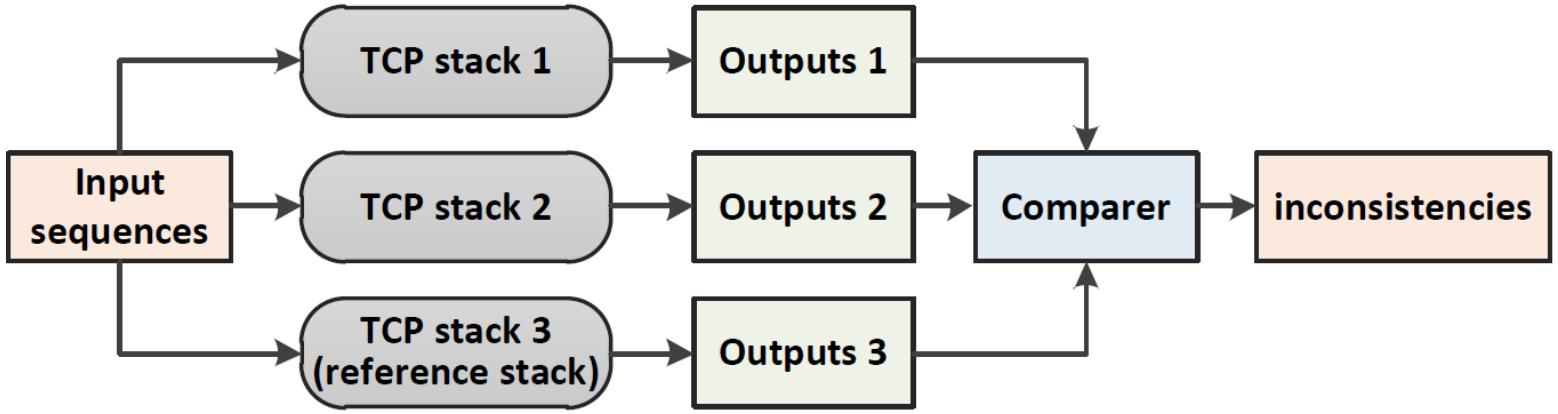
Transition-guided: T3 is new → Input item4 is **interesting**

Transition-Guided Fuzzing Approach



Differential Checker

- Check the output of the same input on different TCP Stacks
  - Outputs are different → at least one has a bugs;
  - Output of the tested TCP stack is different from the standard output → the tested TCP stack has a bug;



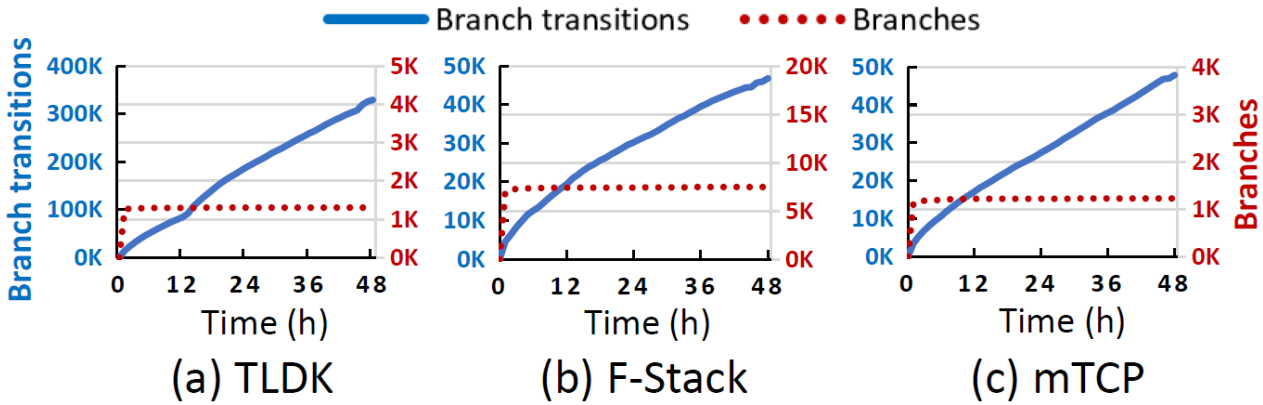
Evaluation

- Test objects
  - user-level
    - TLDK (recent);
    - F-Stack (recent);
    - mTCP (well-known in academic).
  - Kernel-level (test cases generated from the user-level TCP stacks, also used as the reference TCP Stack)
    - FreeBSD TCP (classical);
    - Linux TCP (classical).

Evaluation

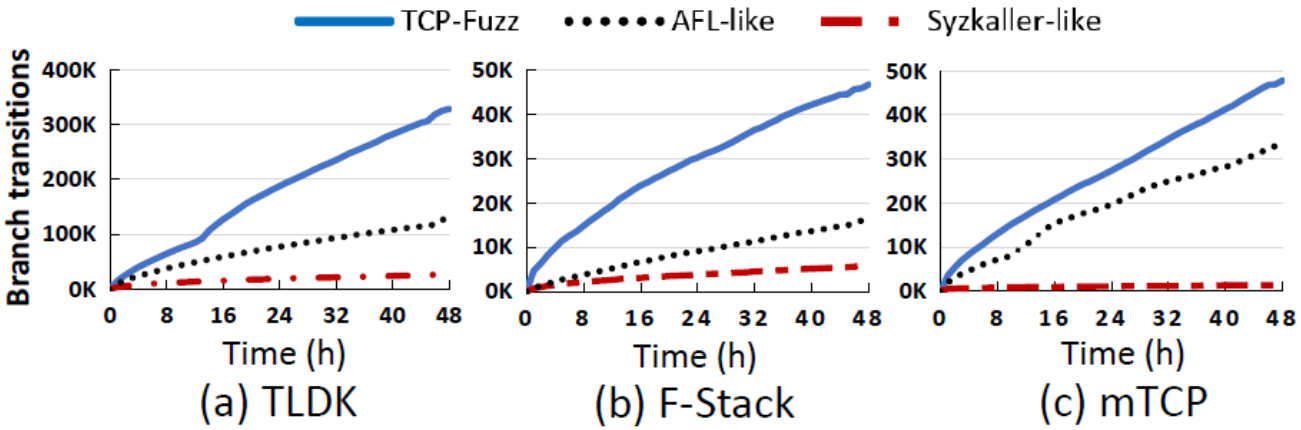
- Results (without comparison)
  - Covers many more branch transitions than branches;
  - Covers few new branches during the later tests.

Stack	Testing coverage		Found bugs	
	Branch	*Transition*	*Memory / Semantic*	*Confirmed / Fixed*
TLDK	1.3K	329.4K	2 / 26	28 / 19
F-Stack	7.5K	46.8K	1 / 6	6 / 1
mTCP	1.2K	47.9K	5 / 9	0 / 0
FreeBSD	-	-	0 / 6	5 / 2
Linux	-	-	0 / 1	1 / 1
Total	10.0K	424.1K	8 / 48	40 / 23

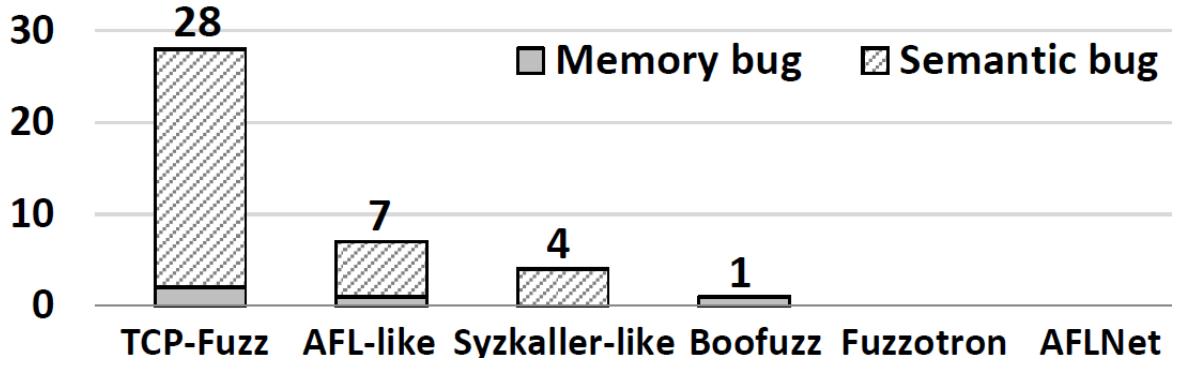


Evaluation

- Compare Objects
  - AFL-like (code coverage);
  - Syzkaller-like (code coverage);
  - Boofuzz, Fuzzotron, AFLNet (open-source protocol fuzzing approaches).
- Results(with comparison, TLDK as example)



In 26 semantic bugs:
25 of these are found by differential checker



Conclusion

- Find problems and solve them one by one
  - **Lack of dependency on input generated by the test;**

    Dependency-based strategy to generate effective test case
  - **Neglect the coverage of state transitions;**

    Transition-guided fuzzing approach to improve the coverage of state transitions
  - **Lack effective detection of semantic bugs.**

    Design **differential checker** to detect semantic bugs