Registered Report: NSFuzz: Towards Efficient and State-Aware Network Service Fuzzing

Shisong Qin[1], Fan Hu[2], Bodong Zhao[1], Tingting Yin[1], Chao Zhang[1]

1. Tsinghua University
2. State Key Laboratory of Mathematical Engineering and Advance

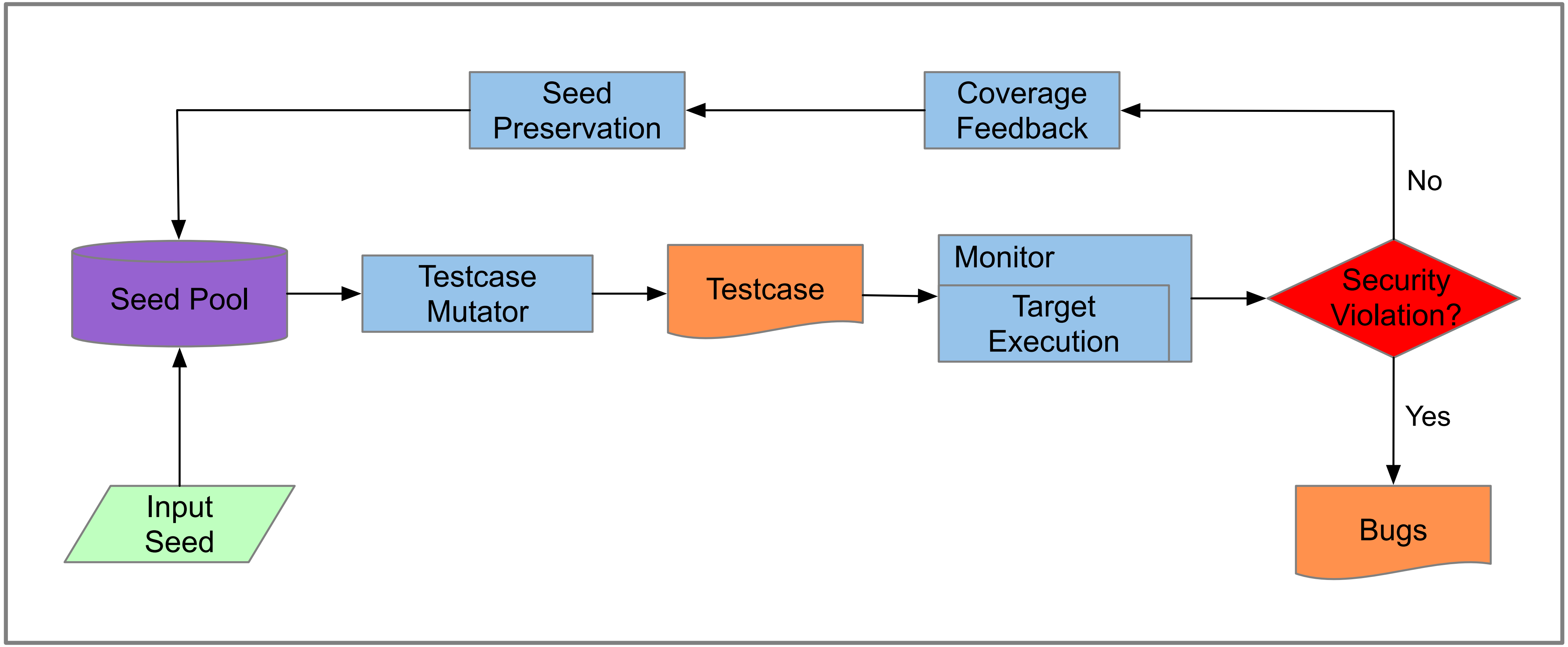
Vulnerability in Network Service

Vulnerabilities in network service enable attackers to **launch remote exploits** much easier than in local applications

Heartbleed from OpenSSL
Remote Confidential Data Leakage

WannaCry from Microsoft's SMB protocol
Ransomware Cyberattack

Fuzzing



The workflow of coverage-guided grey-box fuzzing

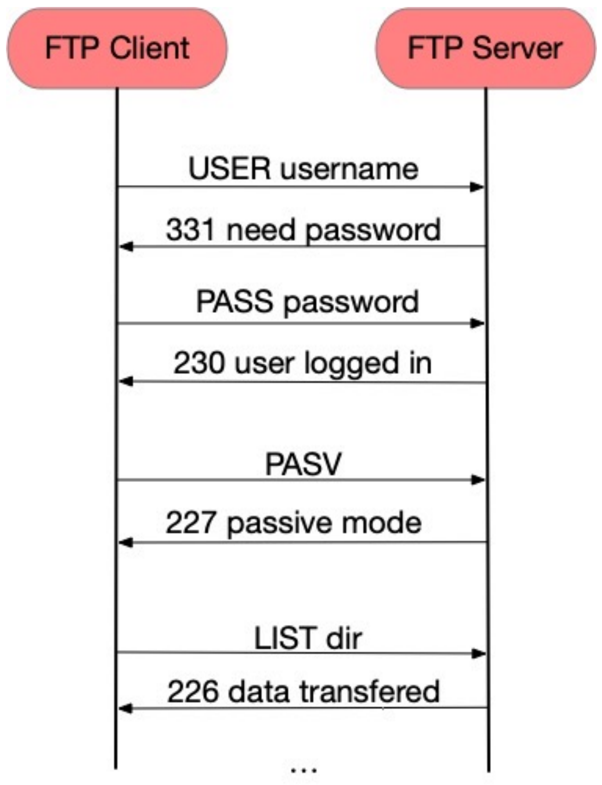
Related Work

Black-Box Network Fuzzing	Grey-Box Network Fuzzing	Program State Model Inference
SPIKE SNOOZE KiF AutoFuzz PULSAR Peach boofuzz	IoTHunter yFuzz SGPFuzz AFLNet StateAFL	Prospex PRETT IJON FuzzFactory AFLNet StateAFL

Have limitations in **fuzzing efficiency** or **service state representation**

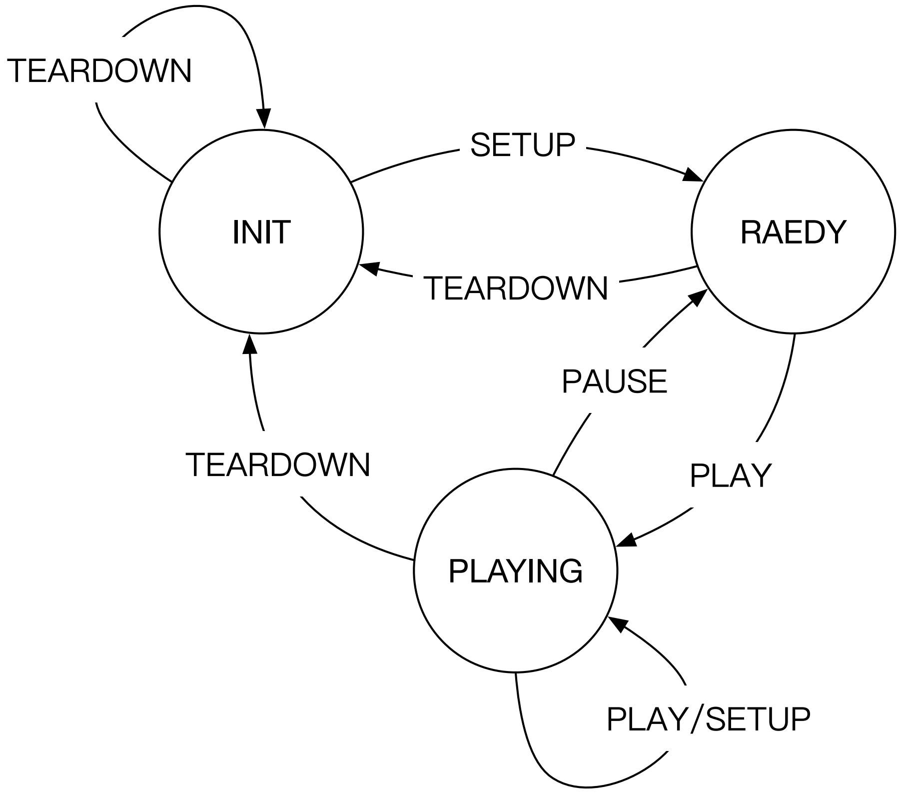
Features of Network Service

Multiple Network I/O Interactions



Multiple interactions between FTP client/server

Involving State Transition (Stateful)

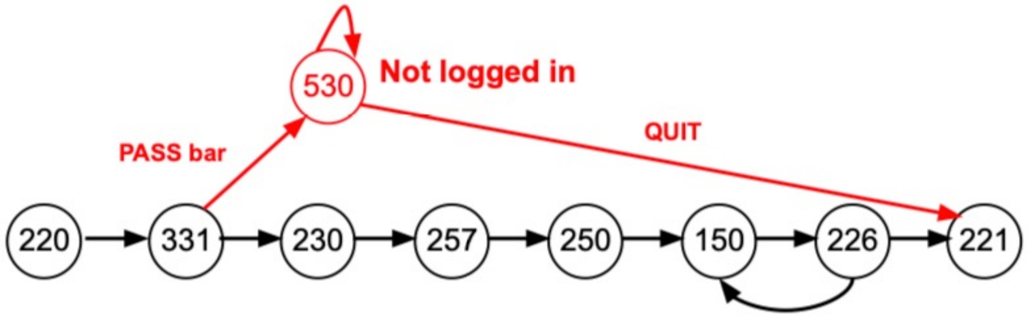


RTSP protocol state model

Challenges in Network Service Fuzzing

- **Service State Representation**
  - Most existing grey-box fuzzers are mainly designed for local stateless applications
  - Fuzzer without state-aware may mislead the evolutionary direction of genetic algorithms due to the stateful of network services

AFLNet[1] : Response code based state representation scheme

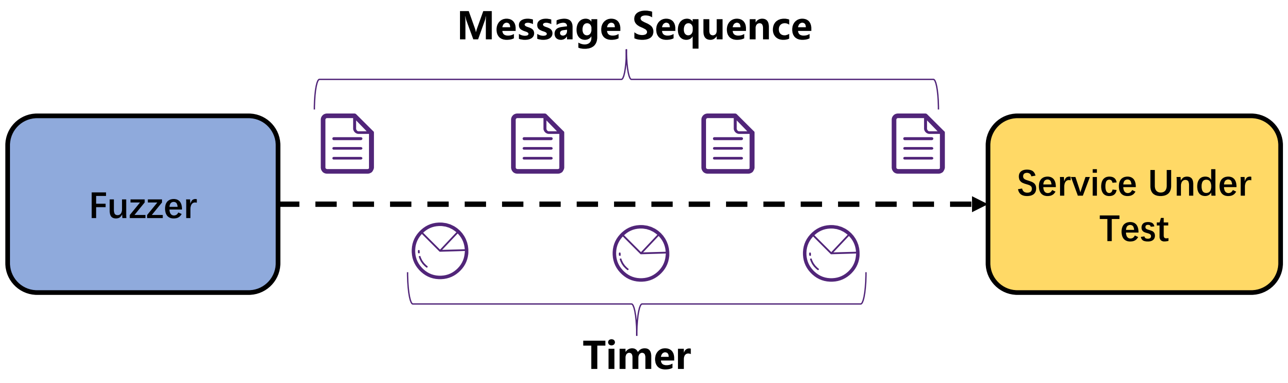


The FTP state model inferred by AFLNet

[1]. Pham V T, Böhme M, et al. AFLNet: A Greybox Fuzzer for Network Protocols. ICST, 2020

Challenges in Network Service Fuzzing

- **Testing Efficiency**
  - Network services are always designed as C/S architecture, requiring multiple I/O
  - Fuzzer needs to conduct multiple interactions to fuzz the service in-depth, and the control of interaction is vital to the fuzzing efficiency



Timer-based I/O interaction control used by AFLNet[1] and StateAFL[2]

[1]. Pham V T, Böhme M, et al. AFLNet: A Greybox Fuzzer for Network Protocols. ICST, 2020
[2]. Roberto Natella. StateAFL: Greybox Fuzzing for Stateful Network Servers

Case Study

```c
int state = STATE_CONNECTED;
int main() {
  ...
  while (fgets(str, MAXCMD, stdin)) { // event loop
    parsecmd(str);
  }
  ...
}
int parsecmd(char *str) {
  ...
  for (i = 0; commands[i].name; i++) {
    if (!strncasecmp(str, commands[i].name, strlen(
     commands[i].name))){
      // state check
      if (state >= commands[i].state_needed) {
        commands[i].function(str); // invoke handler
        return 0;
      } else {
      switch (state) {
      case STATE_CONNECTED:
        response("503 USER expected");
        return 1;
      case STATE_USER:
        response("503 PASS expected");
        return 1;
      case STATE_AUTHENTICATED:
        response("503 RNFR before RNTO expected");
        return 1;}
      }
    }
  }
}
void command_pass(char *password) {
  ...
  if (bftpd_login(password)) {
    state = STATE_CONNECTED; // state update
    return;
  }
}
```

Code snippet from FTP service BFTPD

Case Study

```
int state = STATE_CONNECTED;
int main() {
  ...
  while (fgets(str, MAXCMD, stdin)) { // event loop
    parsecmd(str);
  }
  ...
}
int parsecmd(char *str) {
  ...
  for (i = 0; commands[i].name; i++) {
    if (!strncasecmp(str, commands[i].name, strlen(
     commands[i].name))){
      // state check
      if (state >= commands[i].state_needed) {
        commands[i].function(str); // invoke handler
        return 0;
      } else {
      switch (state) {
      case STATE_CONNECTED:
        response("503 USER expected");
        return 1;
      case STATE_USER:
        response("503 PASS expected");
        return 1;
      case STATE_AUTHENTICATED:
        response("503 RNFR before RNTO expected");
        return 1;}
      }
    }
  }
}
void command_pass(char *password) {
  ...
  if (bftpd_login(password)) {
    state = STATE_CONNECTED; // state update
    return;
  }
}
```

Code snippet from FTP service BFTPD

Network Service

- Use an event loop to perform multiple I/O interactions

Case Study

```c
int state = STATE_CONNECTED;
int main() {
  ...
  while (fgets(str, MAXCMD, stdin)) { // event loop
    parsecmd(str);
  }
  ...
}
int parsecmd(char *str) {
  ...
  for (i = 0; commands[i].name; i++) {
    if (!strncasecmp(str, commands[i].name, strlen(
     commands[i].name))){
      // state check
      if (state >= commands[i].state_needed) {
        commands[i].function(str); // invoke handler
        return 0;
      } else {
      switch (state) {
      case STATE_CONNECTED:
        response("503 USER expected");
        return 1;
      case STATE_USER:
        response("503 PASS expected");
        return 1;
      case STATE_AUTHENTICATED:
        response("503 RNFR before RNTO expected");
        return 1;}
      }
    }
  }
}
void command_pass(char *password) {
  ...
  if (bftpd_login(password)) {
    state = STATE_CONNECTED; // state update
    return;
  }
}
```

Code snippet from FTP service BFTPD

Network Service

- Use an event loop to perform multiple I/O interactions
- Use specific variable to record the current service state

Case Study

```
int state = STATE_CONNECTED;
int main() {
  ...
  while (fgets(str, MAXCMD, stdin)) { // event loop
    parsecmd(str);
  }
  ...
}
int parsecmd(char *str) {
  ...
  for (i = 0; commands[i].name; i++) {
    if (!strncasecmp(str, commands[i].name, strlen(
     commands[i].name))){
      // state check
      if (state >= commands[i].state_needed) {
        commands[i].function(str); // invoke handler
        return 0;
      } else {
      switch (state) {
      case STATE_CONNECTED:
        response("503 USER expected");
        return 1;
      case STATE_USER:
        response("503 PASS expected");
        return 1;
      case STATE_AUTHENTICATED:
        response("503 RNFR before RNTO expected");
        return 1;}
      }
    }
  }
}
void command_pass(char *password) {
  ...
  if (bftpd_login(password)) {
    state = STATE_CONNECTED; // state update
    return;
  }
}
```

Code snippet from FTP service BFTPD

Network Service

- Use an event loop to perform multiple I/O interactions
- Use specific variable to record the current service state
- Execute different code according to current state, and update the state in specific handler

Insights

- **Service State Representation**
  - Network services always use some specific variables to represent the service state directly
  - Such **"state variable"** could represent the service state more accurately and reasonably

- **Testing Efficiency**
  - Network services always have some clear point to indicate the message processing status
  - E.g., the beginning of event loop indicates the previous message has been handled
  - Such **"I/O sync point"** could give fuzzer timely feedback to enable efficient I/O interaction

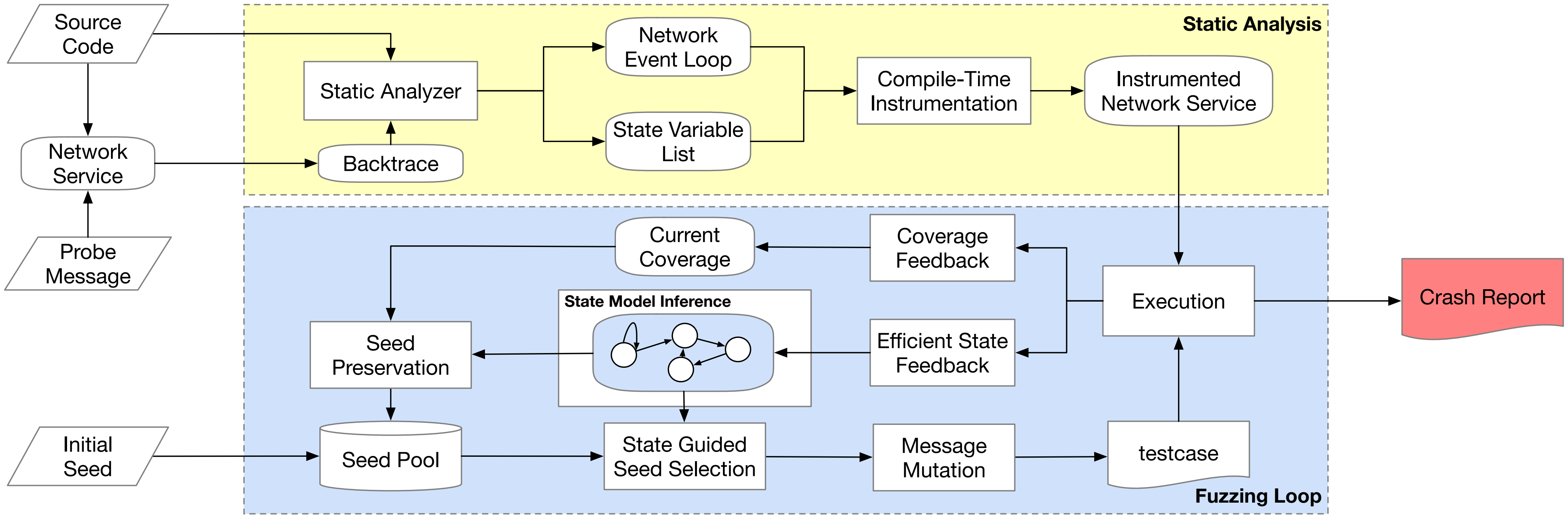
Approach —— NSFuzz

An efficient and state-aware network service fuzzer

- Variable-based accurate service state representation

- Efficient network I/O synchronization mechanism

Overview Design



The workflow of NSFuzz

- Perform static analysis to identify the **event loop (I/O sync point)** and extract **state variables**
- Conduct compile-time instrumentation to enable the target to have the capabilities of signal-based **fast I/O synchronization** and variable-based **service state tracing**
- Carry out efficient and state-aware network service fuzzing loop

Static Analysis

- ## Event Loop Identification

- Use the backtrace of probe message to identify event loop
  - network I/O contained, outermost in the nested loop

- ## State Variable Extraction

- Use a series of heuristic rules to extract state variables
  - range constraint, operation constraint, variable constraint

Compile-Time Instrumentation

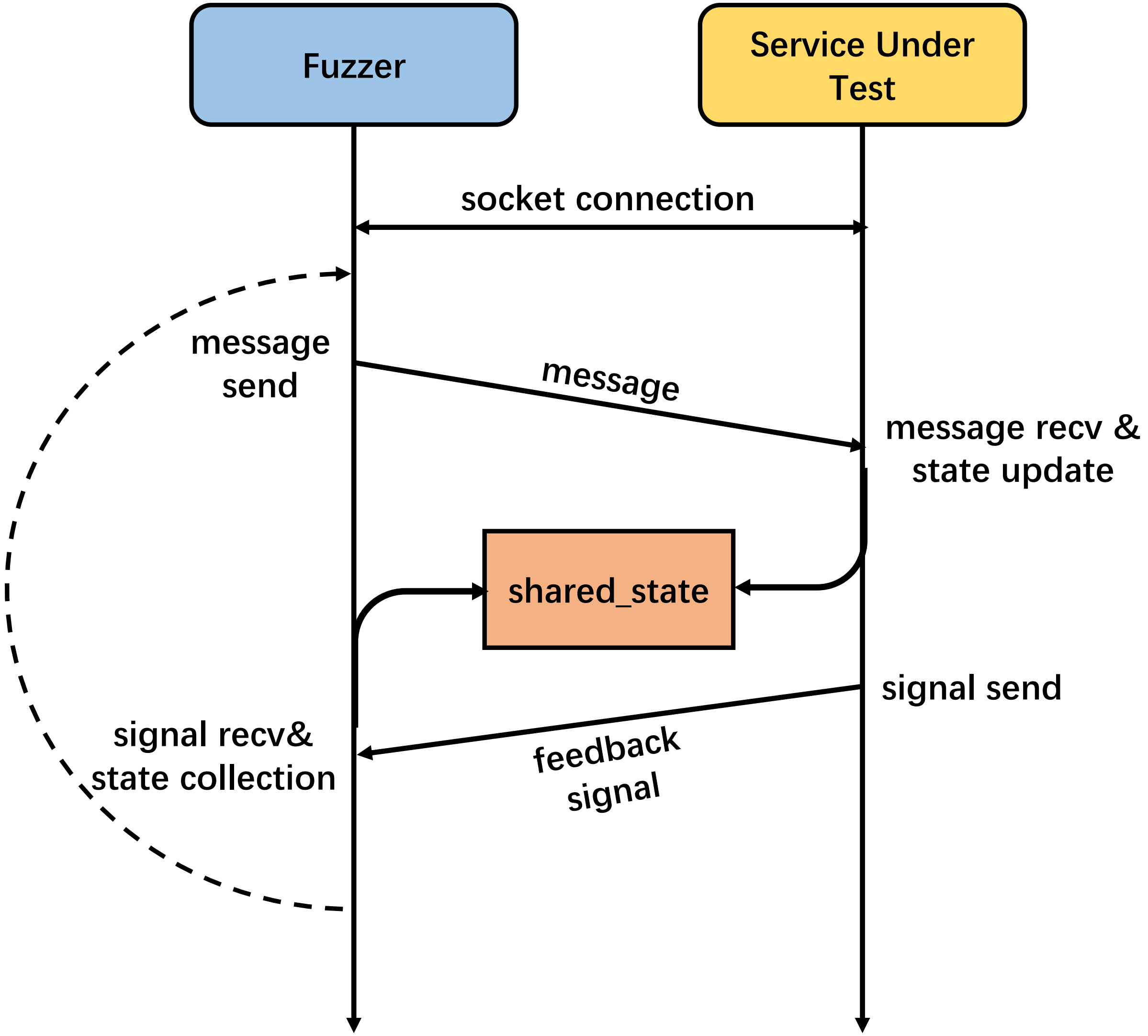
- **Signal Feedback Instrumentation**
  - Insert signal raising function at the **I/O sync point** to give fuzzer feedback
    - *e.g., raise(SIGSTOP)*

Compile-Time Instrumentation

- **Signal Feedback Instrumentation**
  - Insert signal raising function at the **I/O sync point** to give fuzzer feedback
    - *e.g., raise(SIGSTOP)*
- **State Tracing Instrumentation**
  - Setup another shared memory (**shared_state**) between fuzzer and SUT
  - Insert state tracing function at **STORE** operation of each state variable
    - $shared\_state[hash(var_{id}) \oplus cur\_store\_val] = 1$
    - $shared\_state[hash(var_{id}) \oplus pre\_store\_val] = 0$

Fuzzing Loop

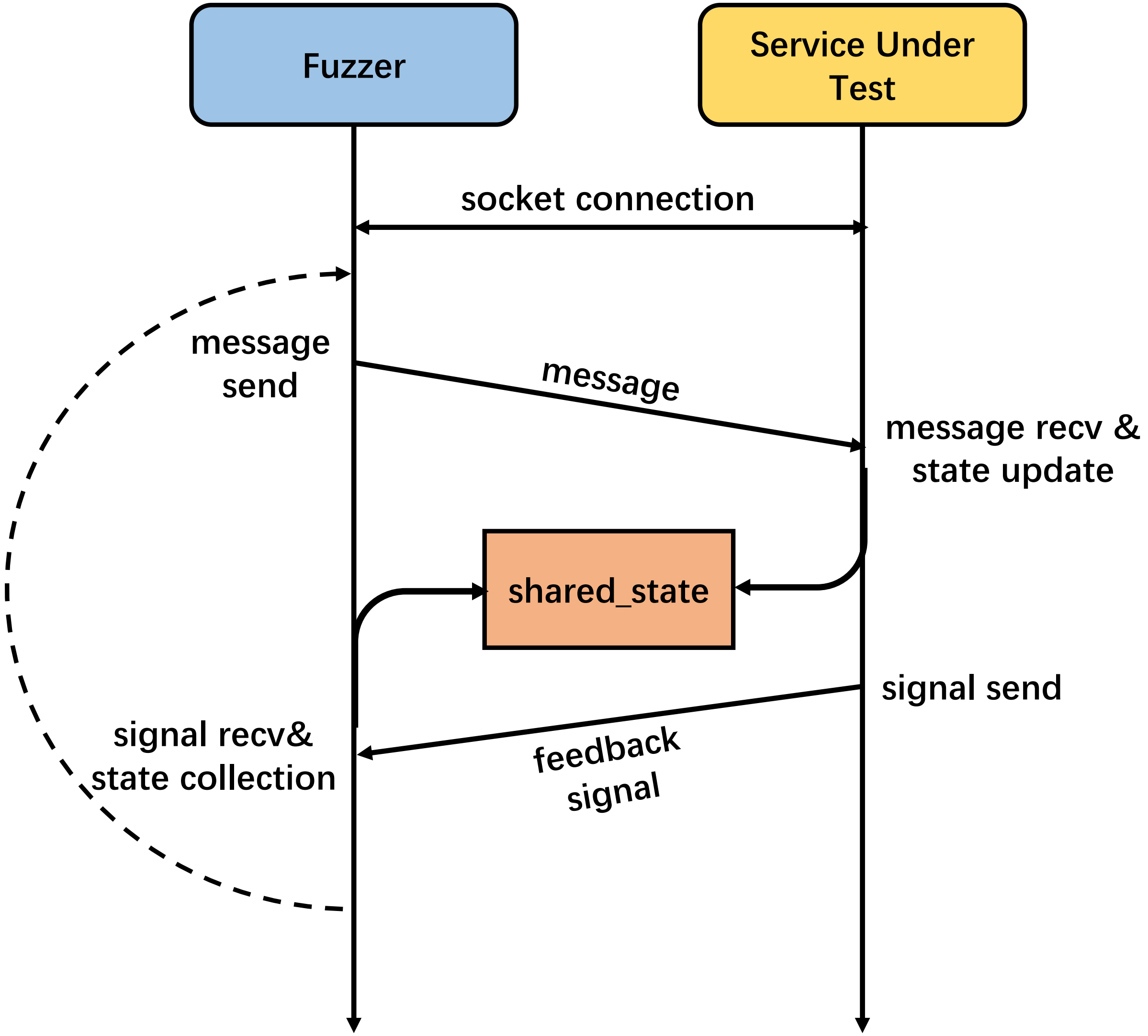
- ➢ **Fast I/O synchronization**



The interaction process between fuzzer and SUT in each testcase

Fuzzing Loop

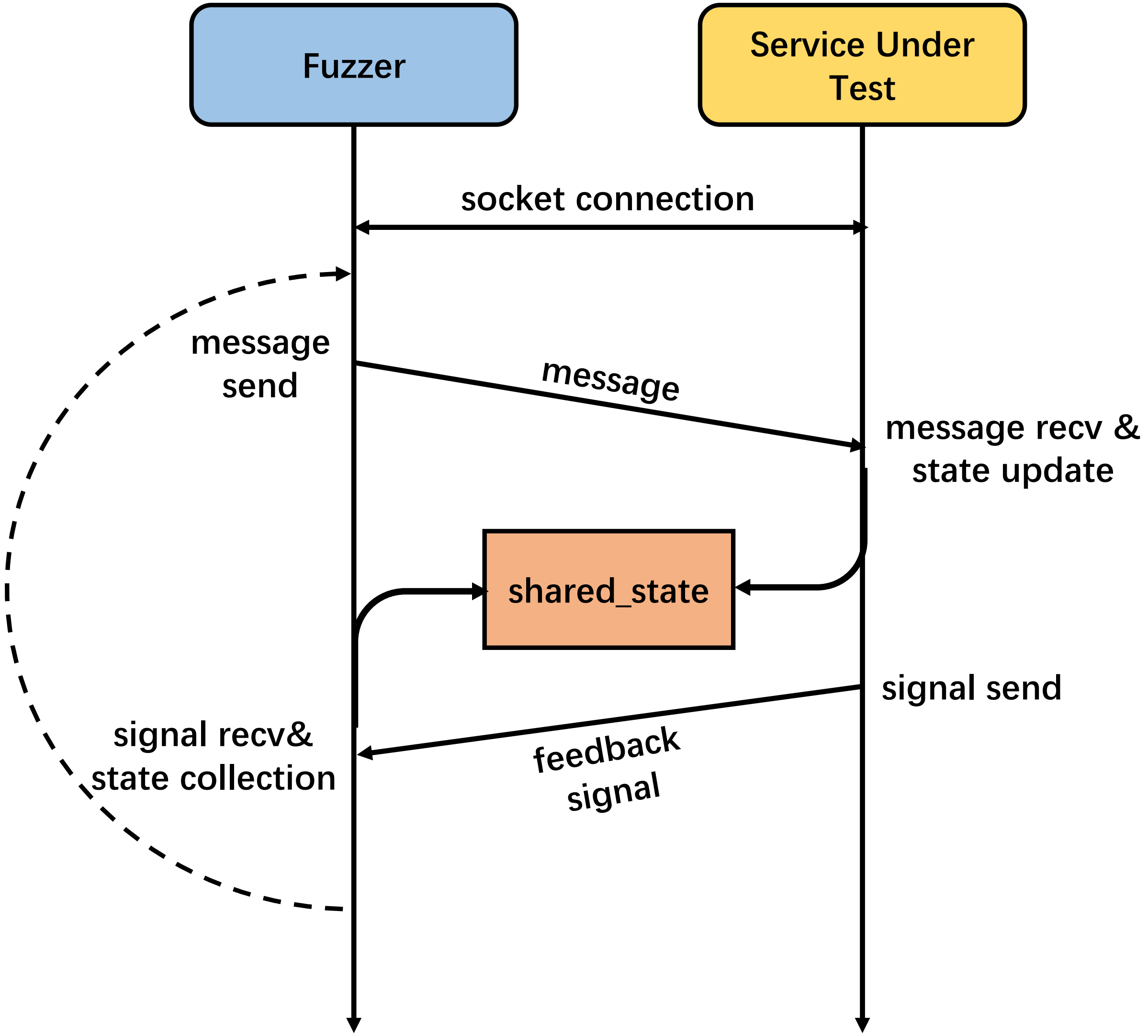
- **Fast I/O synchronization**
  - Each time the fuzzer sends a message, it waits for the signal feedback from service



The interaction process between fuzzer and SUT in each testcase

Fuzzing Loop

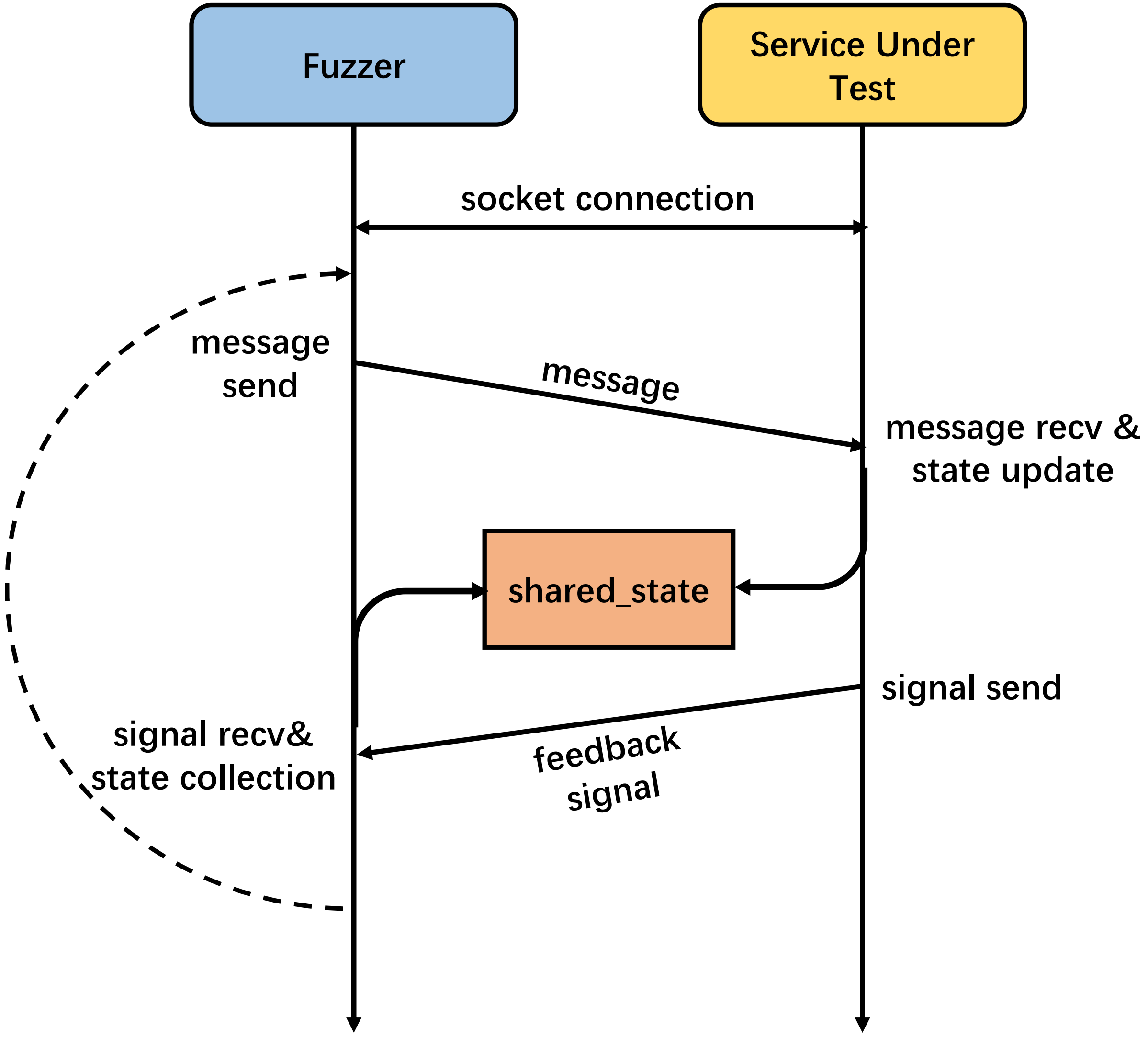
- **Fast I/O synchronization**
  - Each time the fuzzer sends a message, it waits for the signal feedback from service
  - Service receives the message, processes it to update shared_state, then sends a signal



The interaction process between fuzzer and SUT in each testcase

Fuzzing Loop

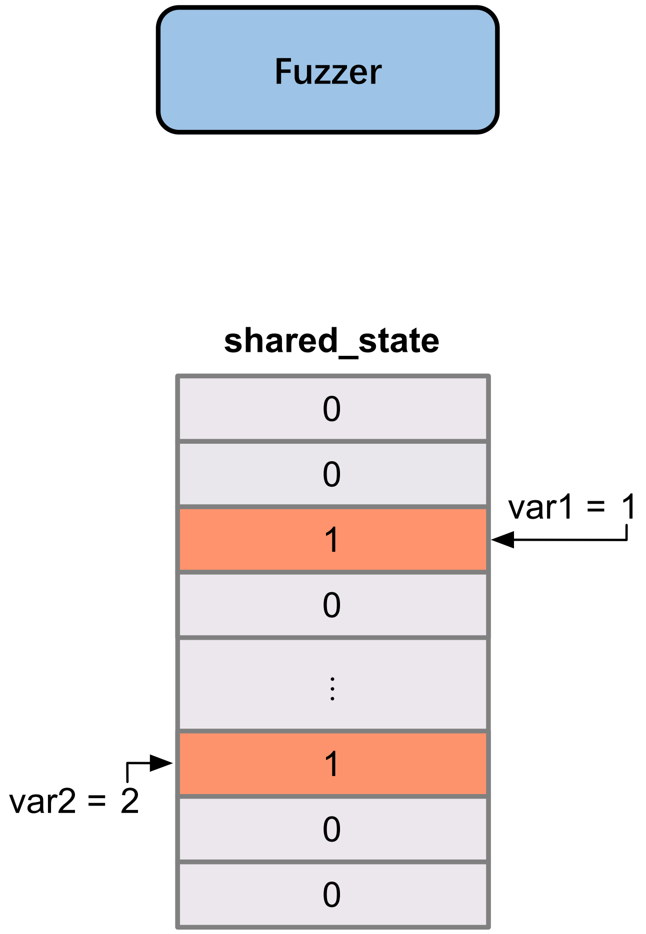
- **Fast I/O synchronization**
  - Each time the fuzzer sends a message, it waits for the signal feedback from service
  - Service receives the message, processes it to update shared_state, then sends a signal
  - Fuzzer receives the signal, collects state representation, then sends the next message



The interaction process between fuzzer and SUT in each testcase

Fuzzing Loop

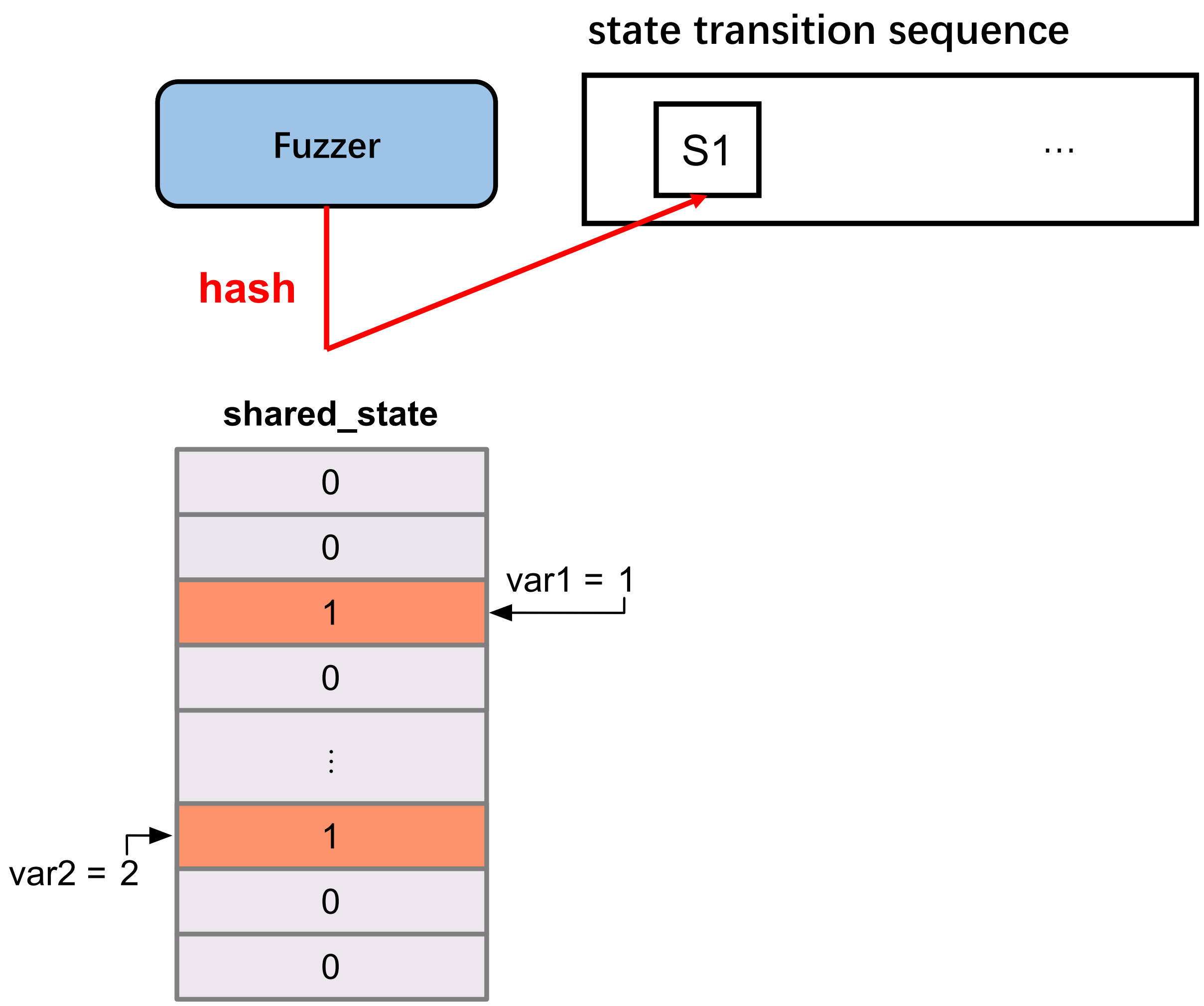
- **Service State Tracing**



The process of shared_state update and state collection

Fuzzing Loop

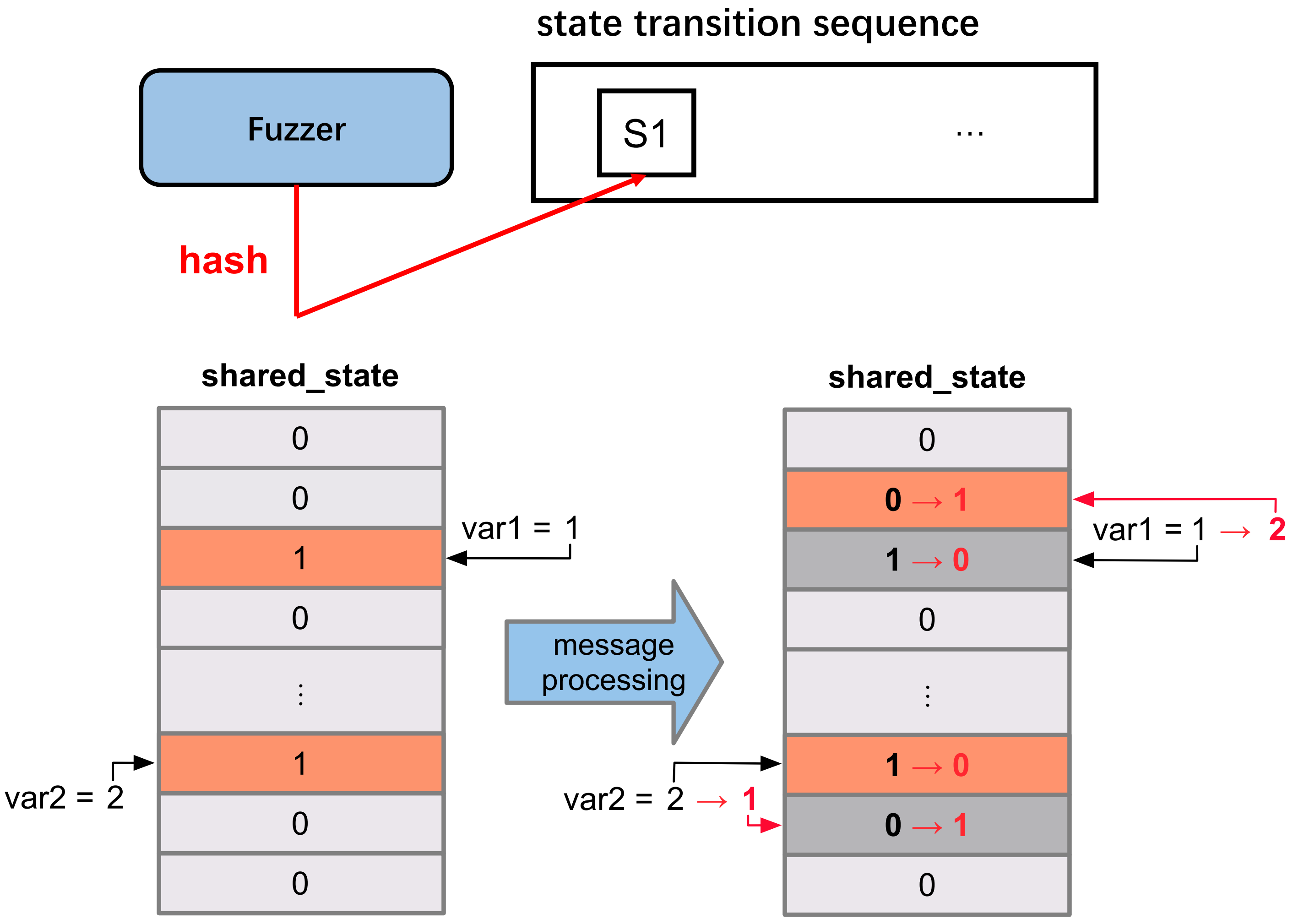
- **Service State Tracing**
  - Fuzzer hash the shared_state to collect state representation when receiving signal feedback



The process of shared_state update and state collection

Fuzzing Loop

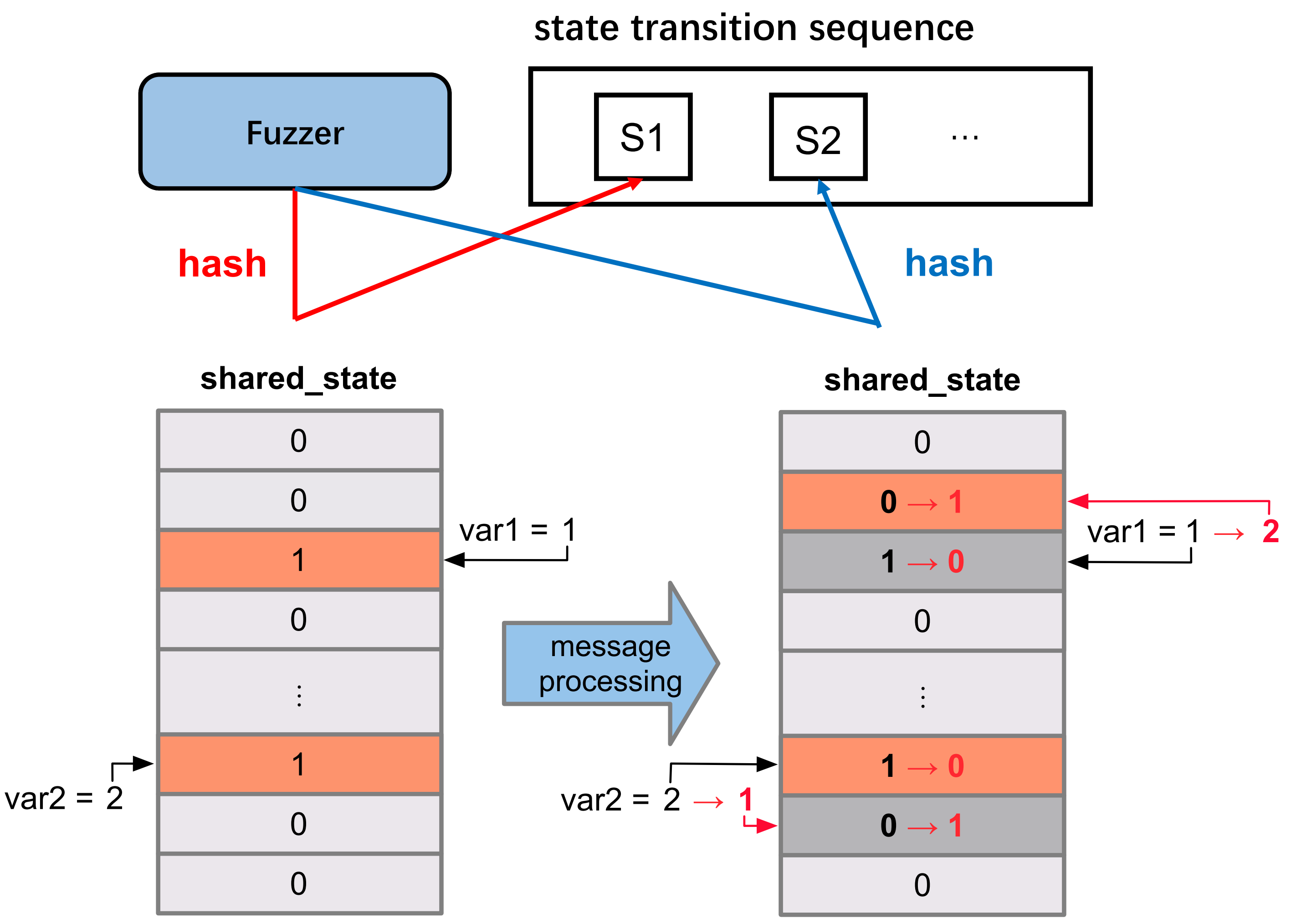
- **Service State Tracing**
  - Fuzzer hash the shared_state to collect state representation when receiving signal feedback
  - A change in any state variable would lead to a change in the hash of shared_state



The process of shared_state update and state collection

Fuzzing Loop

- **Service State Tracing**
  - Fuzzer hash the shared_state to collect state representation when receiving signal feedback
  - A change in any state variable would lead to a change in the hash of shared_state
  - Fuzzer continuous collects state to build transition sequence (model inference)



The process of shared_state update and state collection

Preliminary Evaluation on NSFuzz

- **RQ1: Accurateness of state module inferred by NSFuzz**
  - Could NSFuzz inference relatively more accurate & reasonable state model based on the state variables during the fuzzing loop?

- **RQ2: Effectiveness of NSFuzz state-aware fuzzing**
  - Could NSFuzz achieve higher fuzzing efficiency and overall results than other existing approaches?

Experiment Setup

- 7 targets from ProFuzzBench[1]
- Compared with AFLNet [2] /AFLNwe [3] /StateAFL [4]

Target Service	Network Protocol	Version/Commit	Transport Layer	Language
LightFTP	FTP	5980ea1	TCP	C
Bftpd	FTP	v5.7	TCP	C
Pure-FTPd	FTP	c21b45f	TCP	C
Exim	SMTP	38903fb	TCP	C
Dnsmasq	DNS	v2.73rc6	UDP	C
TinyDTLS	DTLS	06995d4	UDP	C
Kamailio	SIP	2648eb3	UDP	C

The selected evaluation target

[1]. https://github.com/profuzzbench/profuzzbench
[2]. https://github.com/profuzzbench/aflnet
[3]. https://github.com/profuzzbench/aflnwe
[4]. https://github.com/stateafl/stateafl

State Module Inference Evaluation (RQ1)

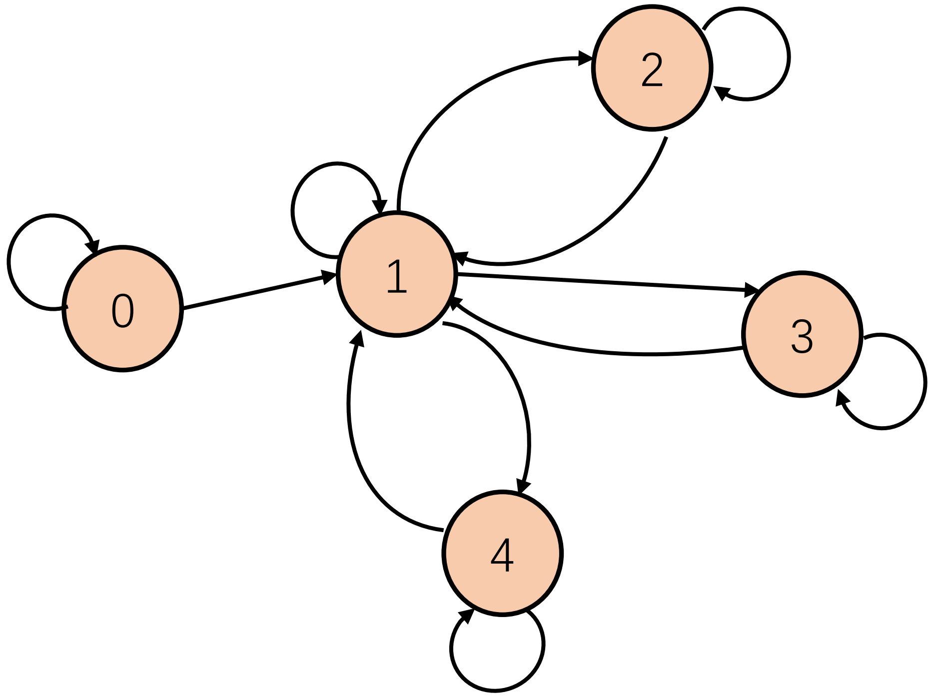
Target Service	LoC	Network Event Loop	State Variable		Analysis Time
			Number	Example	
LightFTP	4.4k	√	1	Access	0.7s
Bftpd	4.7k	√	6	state	1.8s
Pure-FTPd	30k	√	22	loggedin	3.9s
Exim	101.7k	√	58	helo_seen	45.1s
Dnsmasq	27.6k	√	15	found	11.4s
TinyDTLS	10.8k	√	4	state	3.2s
Kamailio	766.7k	√	58	state	441.9s

The static analysis results on evaluation target

State Module Inference Evaluation (RQ1)

Target Service	Fuzzer	State Module	
		Vertexes	Edges
LightFTP	AFLNET	23	158
	STATEAFL	11	47
	NSFuzz	5	12
Bftpd	AFLNET	24	126
	STATEAFL	4	6
	NSFuzz	43	137
Pure-FTPd	AFLNET	27	260
	STATEAFL	7	22
	NSFuzz	8	22
Exim	AFLNET	12	60
	STATEAFL	7	17
	NSFuzz	128	225
Dnsmasq	AFLNET	89	271
	STATEAFL	108	467
	NSFuzz	3	5
TinyDTLS	AFLNET	9	24
	STATEAFL	29	69
	NSFuzz	32	115
Kamailio	AFLNET	13	93
	STATEAFL	4	4
	NSFuzz	99	328

The state model inferred by various fuzzers



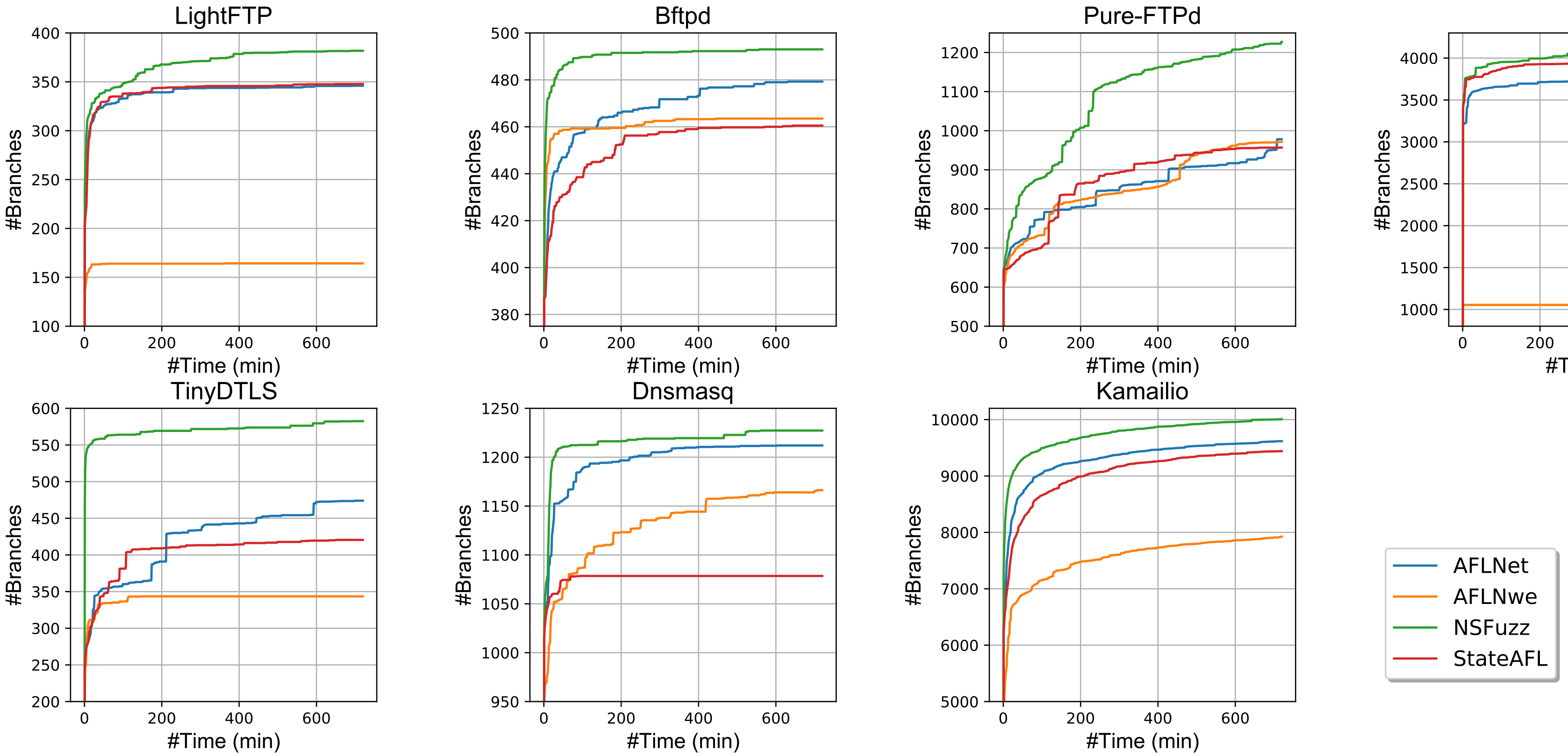
The state model of LightFTP inferred by NSFuzz

Fuzzing Efficiency Evaluation (RQ2)

Target Service	Fuzzing Throughput (exec/s)			
	AFLNet	AFLNwe	StateAFL	NSFuzz
LightFTP	8.42	+330.8%	-55.6%	**+558.9%**
Bftpd	4.09	+144.0%	-45.2%	**+869.7%**
Pure-FTPd	5.29	+115.3%	-80.0%	**+175.0%**
Exim	2.69	+108.6%	+35.3%	**+113.4%**
Dnsmasq	7.47	+454.2%	-82.7%	**+645.1%**
TinyDTLS	2.66	+458.3%	-47.0%	**+5488.0%**
Kamailio	5.19	+20.8%	-49.7%	**+512.5%**

The average fuzzing throughput of various fuzzers toward each target service

Fuzzing Efficiency Evaluation (RQ2)



The average branch coverage growth in 12h of various fuzzers toward each target service

Fuzzing Efficiency Evaluation (RQ2)

Target Service	**Crash Trigger Time (s)**			
	AFLNet	**AFLNwe**	**StateAFL**	**NSFuzz**
Dnsmasq	990.5s	989.25s	878.75s	**160s**
TinyDTLS	26s	11.75s	47.75s	**< 1s**

The average crash trigger time of various fuzzers toward each target service

Limitations

- **Scalability**
  - **Service Pattern** Support (libevent-based target)
  - **Service Language** Support (other than C)
  - **False Positive** in state variable extraction (leading to state explosion)

The fragile of **Static Analysis** is the main reason (e.g., ad-hoc analysis rules…)

Conclusion

- Analyzed the **state representation** and **testing efficiency** challenges of network service fuzzing

- Proposed NSFuzz, a network service fuzzer combined with variable-based state representation and efficient I/O synchronization

- Preliminary evaluated NSFuzz on ProFuzzBench, and the results showed NSFuzz could infer a accurate state model and achieve a higher fuzzing efficiency than some other existing solutions

Ongoing Work

- **Annotation API**
  - **I/O Sync Point Annotation**
    - Multiple I/O point supported
    - libevent-based target supported
  - **State Variable Annotation**
    - Eliminate false positive
    - Precise annotation

I/O sync point annotation usage demo

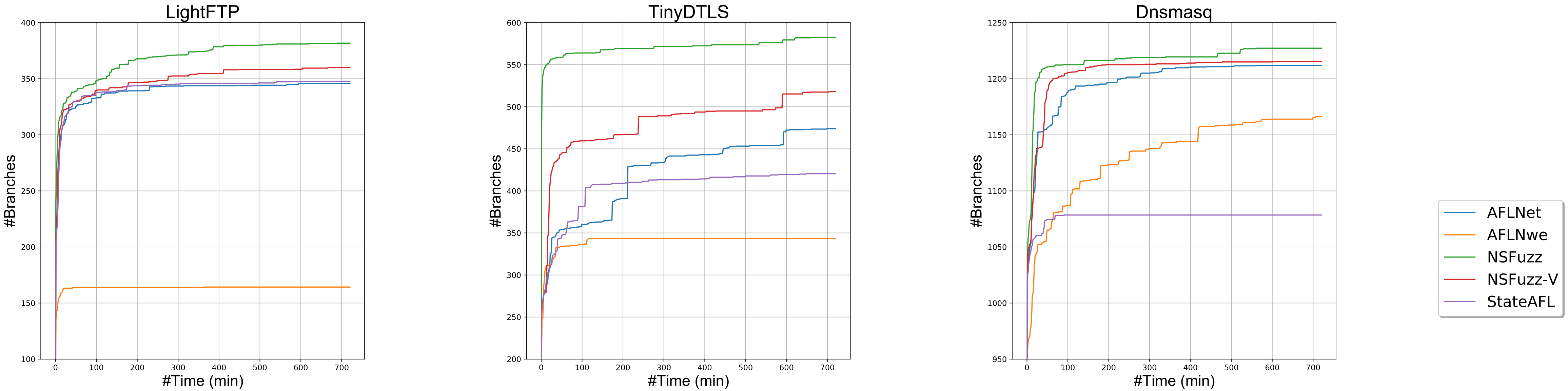
```
int main(int argc, char **argv) {
   ... // service initialization
  while (fgets(str, MAXCMD, sock)) {
    // I/O sync point annotation API
    _NSFUZZ_SYNC();
    ...
    parsecmd(str);
  }
  ...
  return 0;
}
```

state variable annotation usage demo

```
enum {
    STATE_CONNECTED, STATE_USER,
    STATE_AUTHENTICATED, STATE_RENAME, STATE_ADMIN
};
// state variable annotation API (global variable)
int _NSFUZZ_STATE(state) = STATE_CONNECTED;
```

Ongoing Work

➢ Ablation Study



The average branch coverage growth in 12h of various fuzzers toward each target service

NSFuzz-V: NSFuzz with variable-based state representation only enabled

Thanks for Listening!

Q & A

Contact: qss19@mails.tsinghua.edu.cn