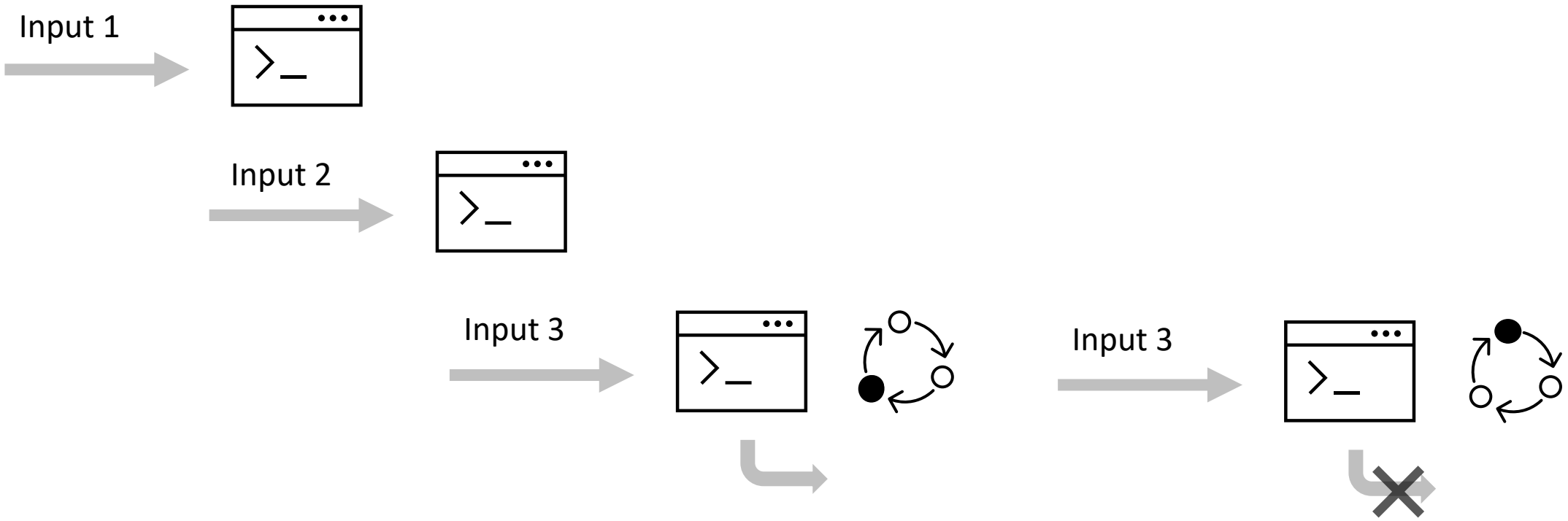
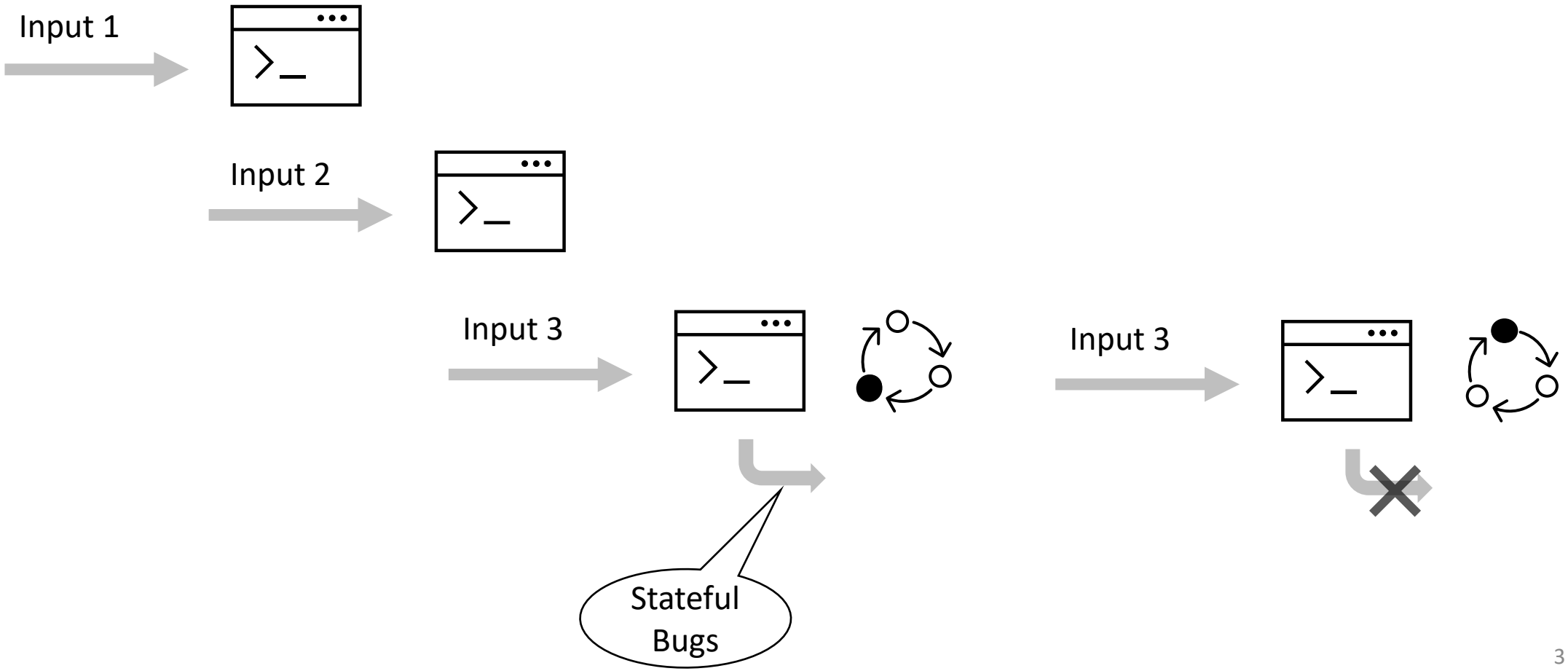
Stateful Greybox Fuzzing

Jinsheng Ba[1], Marcel Böhme[2,3], Zahra Mirzamomen[2], and Abhik Roychoudhury[1]
[1]National University of Singapore, [2]Monash University, [3]MPI-SP

Challenge: Bugs in Stateful Programs

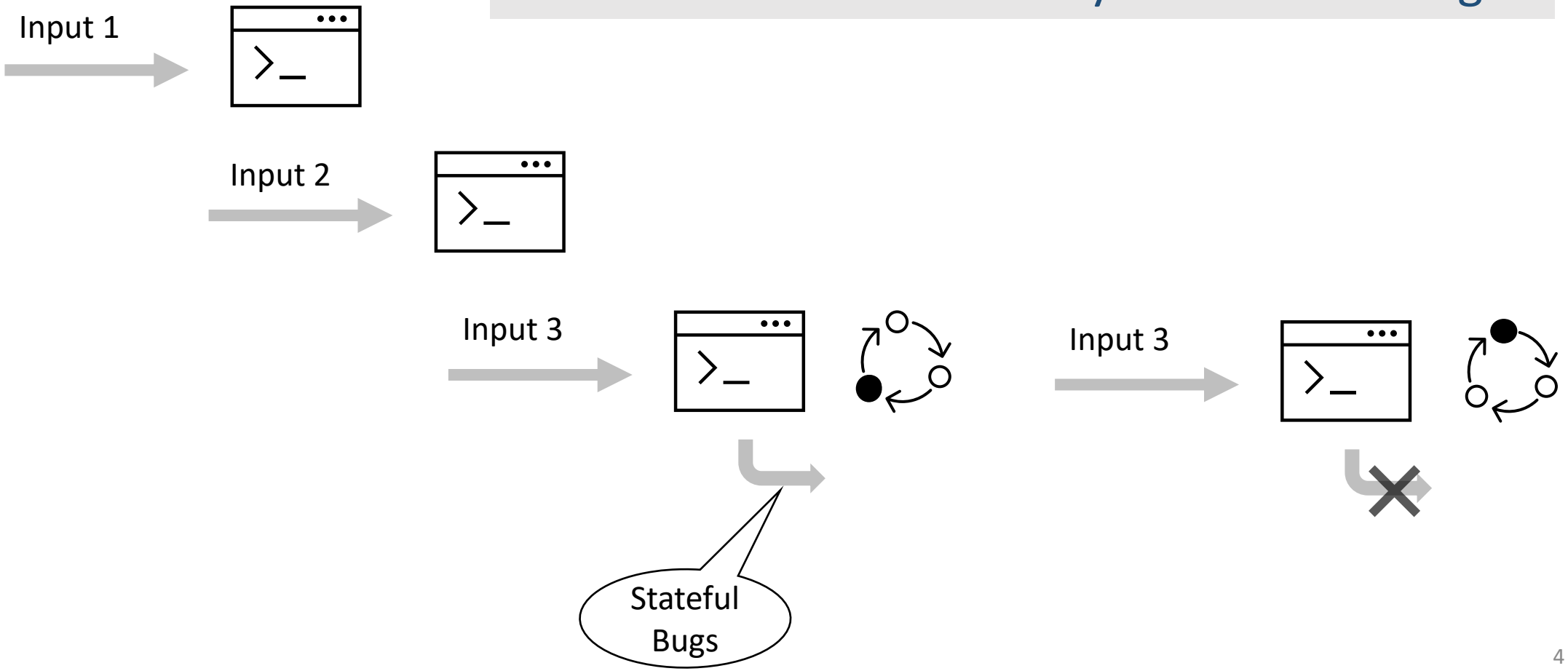


Challenge: Bugs in Stateful Programs



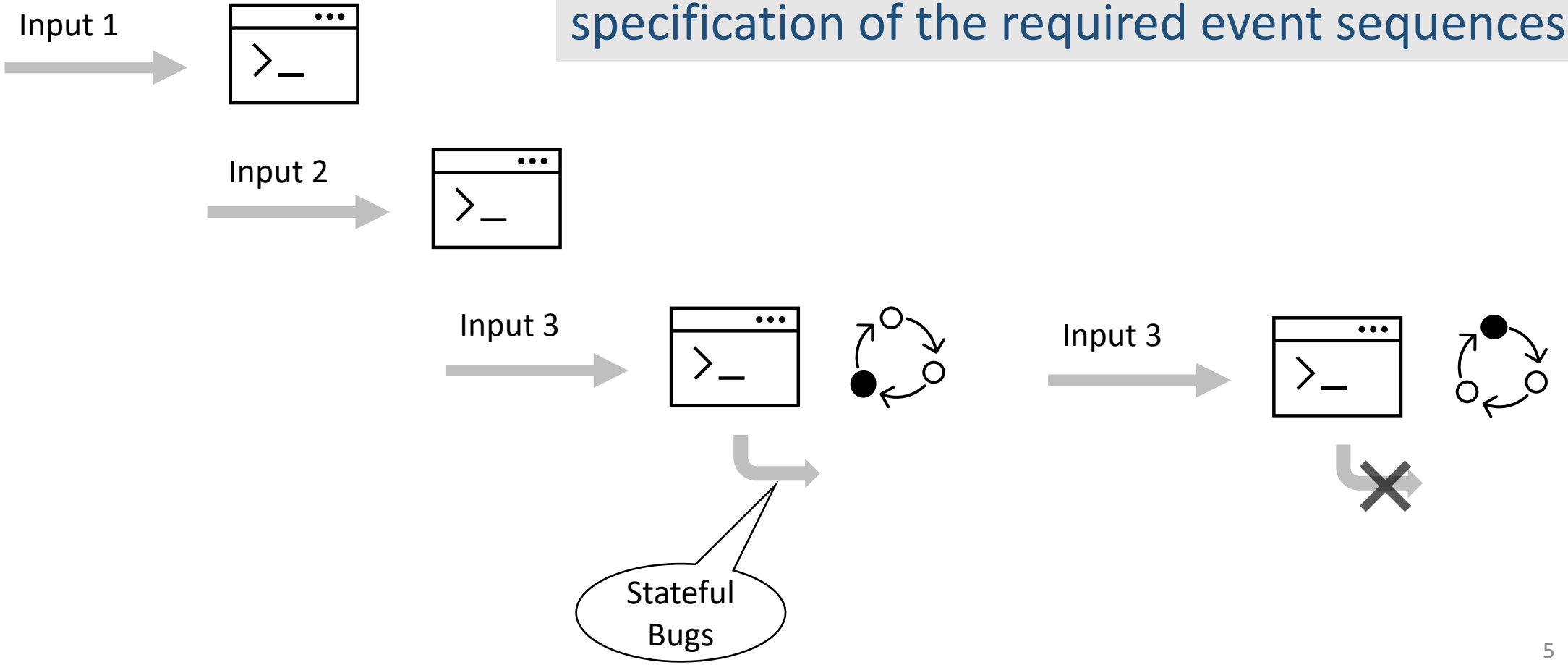
Challenge: Bugs in Stateful Programs

Problem: How to efficiently find stateful bugs?



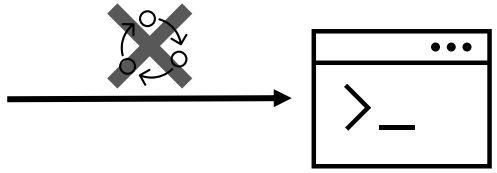
Challenge: Bugs in Stateful Programs

Challenge: Cover the state space without a specification of the required event sequences.



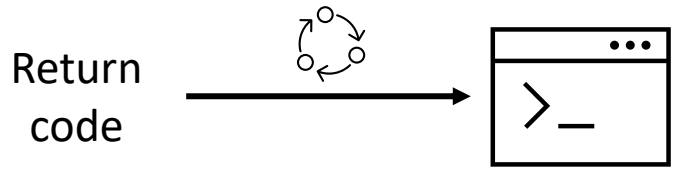
Past Works

- AFL, LibFuzzer



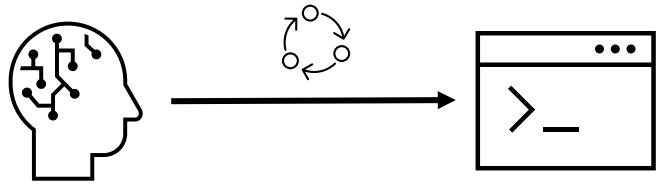
Stateless fuzzers that cannot generate a sequence of inputs in specific orders.

- AFLNet



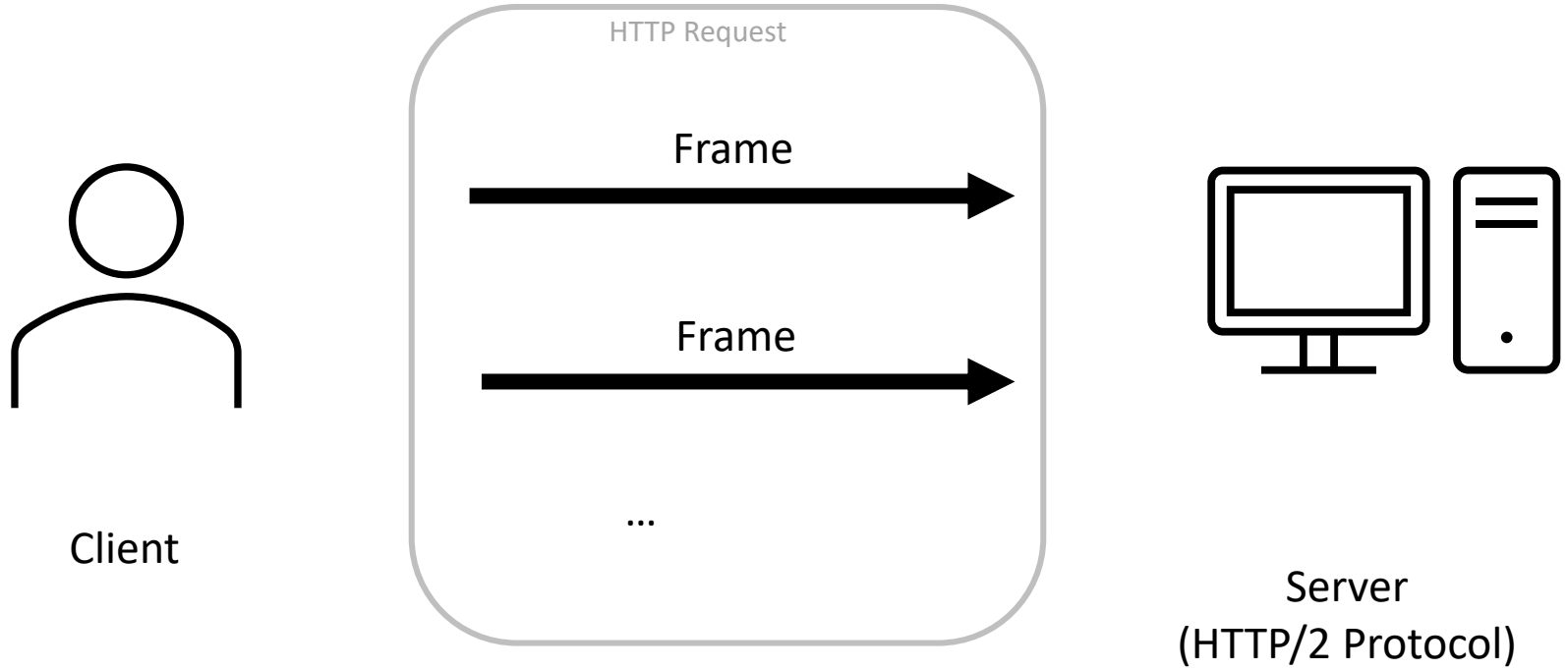
Can not represent program internal states

- IJON

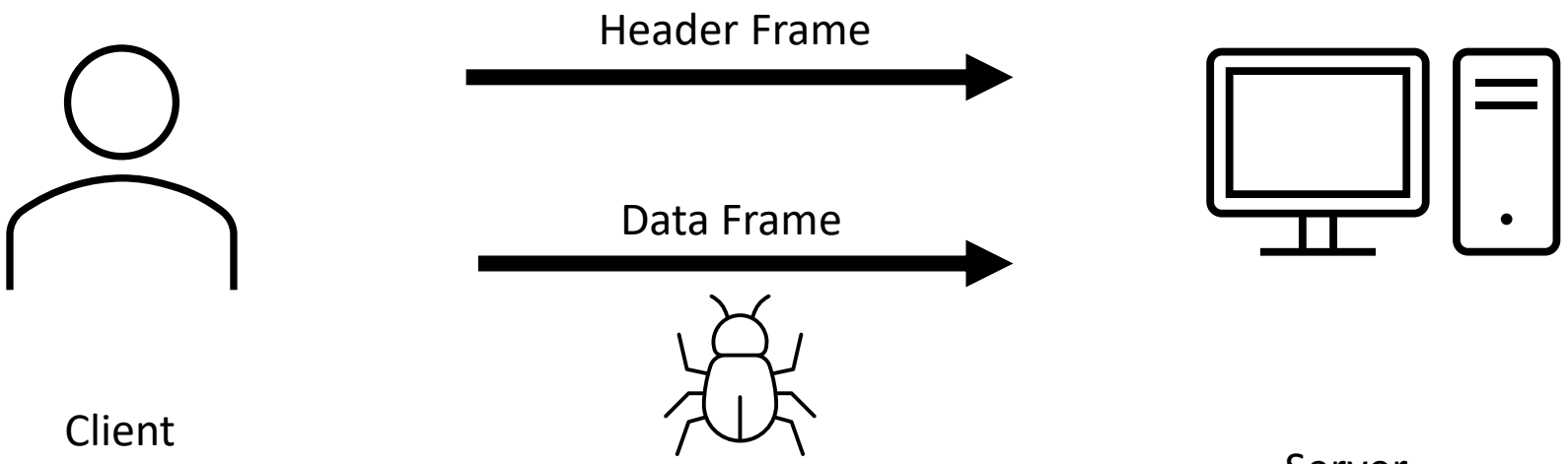


Requires human knowledge (state specification)

Protocol Implementations (Stateful Programs)



Protocol Implementations (Stateful Programs)



State Identification

- They are represented by state variables with named constants.

State Identification

- They are represented by state variables with named constants.

H2O

```
typedef enum enum_h2o_http2_stream_state_t {
    /**
     * stream in idle state (but registered; i.e. priority stream)
     */
    H2O_HTTP2_STREAM_STATE_IDLE,
    /**
     * receiving headers
     */
    H2O_HTTP2_STREAM_STATE_RECV_HEADERS,
    /**
     * receiving body (or trailers), waiting for the arrival of END_STREAM
     */
    H2O_HTTP2_STREAM_STATE_RECV_BODY,
    /**
     * received request but haven't been assigned a handler
     */
    H2O_HTTP2_STREAM_STATE_REQ_PENDING,
```

OpenSSL

```
typedef enum {
    TLS_ST_BEFORE,
    TLS_ST_OK,
    DTLS_ST_CR_HELLO_VERIFY_REQUEST,
    TLS_ST_CR_SRVR_HELLO,
    TLS_ST_CR_CERT,
    TLS_ST_CR_CERT_STATUS,
    TLS_ST_CR_KEY_EXCH,
    TLS_ST_CR_CERT_REQ,
    TLS_ST_CR_SRVR_DONE,
    TLS_ST_CR_SESSION_TICKET,
    TLS_ST_CR_CHANGE,
    TLS_ST_CR_FINISHED,
    TLS_ST_CW_CLNT_HELLO,
    TLS_ST_CW_CERT,
    TLS_ST_CW_KEY_EXCH,
    TLS_ST_CW_CERT_VRFY,
    TLS_ST_CW_CHANGE,
    TLS_ST_CW_NEXT_PROTO,
```

Top-50 most widely used protocol implementations use named constants to represent protocol states.

44 of 50 use enumeration type, 6 use #define macro, to define the named constants.
Including 16 protocols: FTP, SFTP, TLS, SMTP, HTTP2, RDP, NTP, IMAP, IRC, SMB, DAAP, SIP, DICOM, VNC, RTSP, MQTT.

```
static void handle_request_body_chunk(                                  4
   ...
   if (stream->req.write_req.cb(stream->req.write_req.ctx, payload,
       is_end_stream) != 0) {
       ...
}
```

```
static int handle_data_frame(...h2o_http2_stream_t *stream...){         3
    ...
    if (stream->state != H2O_HTTP2_STREAM_STATE_RECV_BODY && ...) {
      ...
      return 0;
    }
    handle_request_body_chunk(...);
    ...
}
```

```
static int handle_incoming_request(...h2o_http2_stream_t *stream...){   2
    ...
    h2o_http2_stream_set_state(conn, stream,
                               H2O_HTTP2_STREAM_STATE_RECV_BODY);
    ...
}
```

```
static int handle_headers_frame(...h2o_http2_frame_t *frame...)         1
{
    ...
    if ((frame->flags & H2O_HTTP2_FRAME_FLAG_END_HEADERS) != 0) {
        /* request headers are complete, handle it */
        return handle_incoming_request(conn, stream, ...);
    }
    ...
    return 0;
}
```

- stream->state

```
static void handle_request_body_chunk(
   ...
   if (stream->req.write_req.cb(stream->req.write_req.ctx, payload,
       is_end_stream) != 0) {
       ...
}
```
④

```
static int handle_data_frame(...h2o_http2_stream_t *stream...){
    ...
    if (stream->state != H2O_HTTP2_STREAM_STATE_RECV_BODY && ...) {
      ...
      return 0;
    }
    handle_request_body_chunk(...);
    ...
}
```
③

```
static int handle_incoming_request(...h2o_http2_stream_t *stream...){
    ...
    h2o_http2_stream_set_state(conn, stream,
                               H2O_HTTP2_STREAM_STATE_RECV_BODY);
    ...
}
```
②

```
static int handle_headers_frame(...h2o_http2_frame_t *frame...)
{
    ...
    if ((frame->flags & H2O_HTTP2_FRAME_FLAG_END_HEADERS) != 0) {
        /* request headers are complete, handle it */
        return handle_incoming_request(conn, stream, ...);
    }
    ...
    return 0;
}
```
①

- stream->state

Data Frame

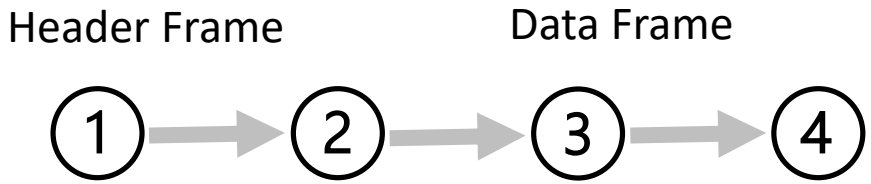
```
static void handle_request_body_chunk(                                  ④
   ...
   if (stream->req.write_req.cb(stream->req.write_req.ctx, payload,
       is_end_stream) != 0) {
       ...
}
```

```
static int handle_data_frame(...h2o_http2_stream_t *stream...){         ③
    ...
    if (stream->state != H2O_HTTP2_STREAM_STATE_RECV_BODY && ...) {
      ...
      return 0;
    }
    handle_request_body_chunk(...);
    ...
}
```

```
static int handle_incoming_request(...h2o_http2_stream_t *stream...){   ②
    ...
    h2o_http2_stream_set_state(conn, stream,
                               H2O_HTTP2_STREAM_STATE_RECV_BODY);
    ...
}
```

```
static int handle_headers_frame(...h2o_http2_frame_t *frame...)         ①
{
    ...
    if ((frame->flags & H2O_HTTP2_FRAME_FLAG_END_HEADERS) != 0) {
        /* request headers are complete, handle it */
        return handle_incoming_request(conn, stream, ...);
    }
    ...
    return 0;
}
```

- stream->state

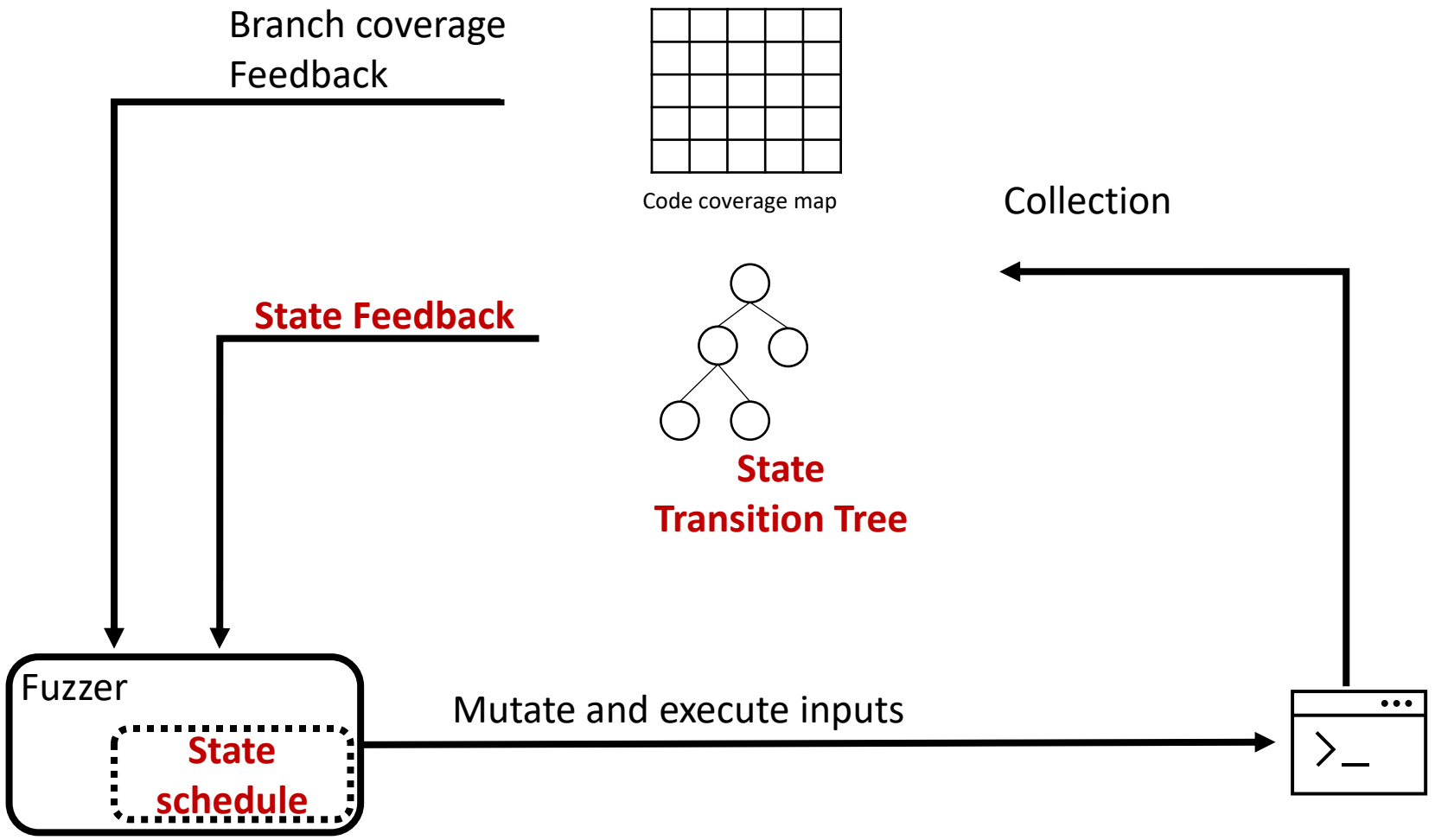


Insight

Approximating state variables by the variables with named constants.

- There may be variables take named constants that are not state variables (e.g., configuration variables, error code variables)
- In our evaluation, over 99% of extracted variables are true states.

Stateful Greybox Fuzzing

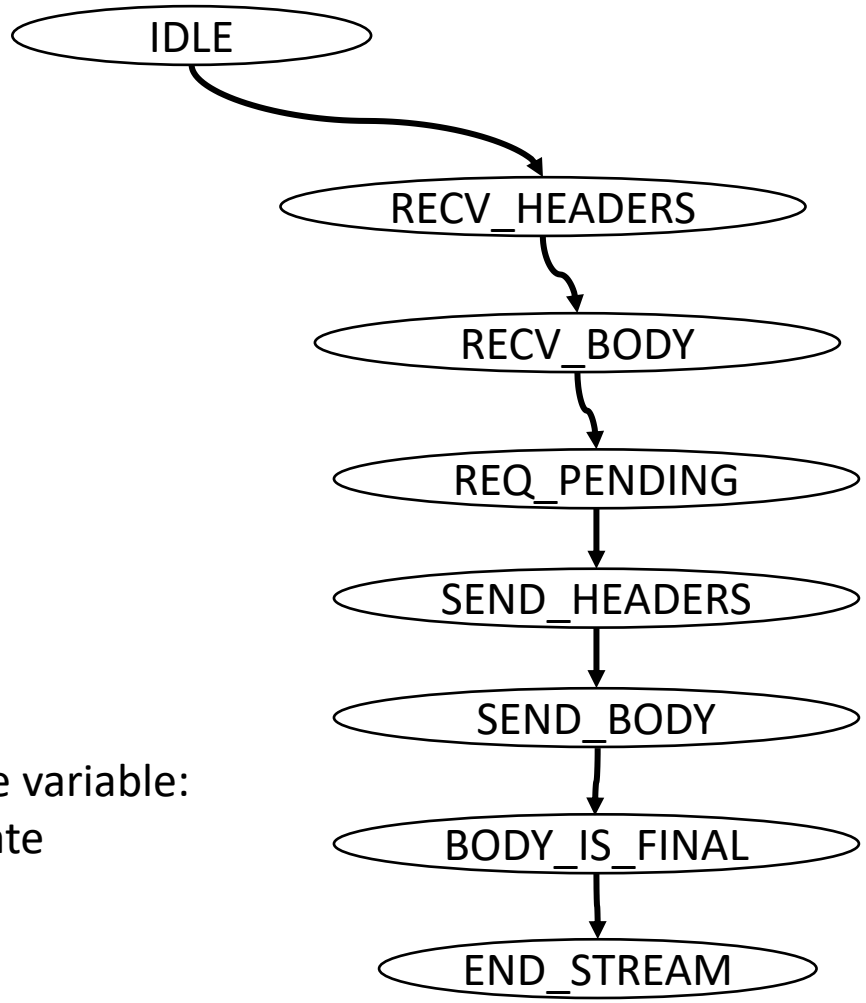


State Transition Tree Construction

In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable:
stream->state

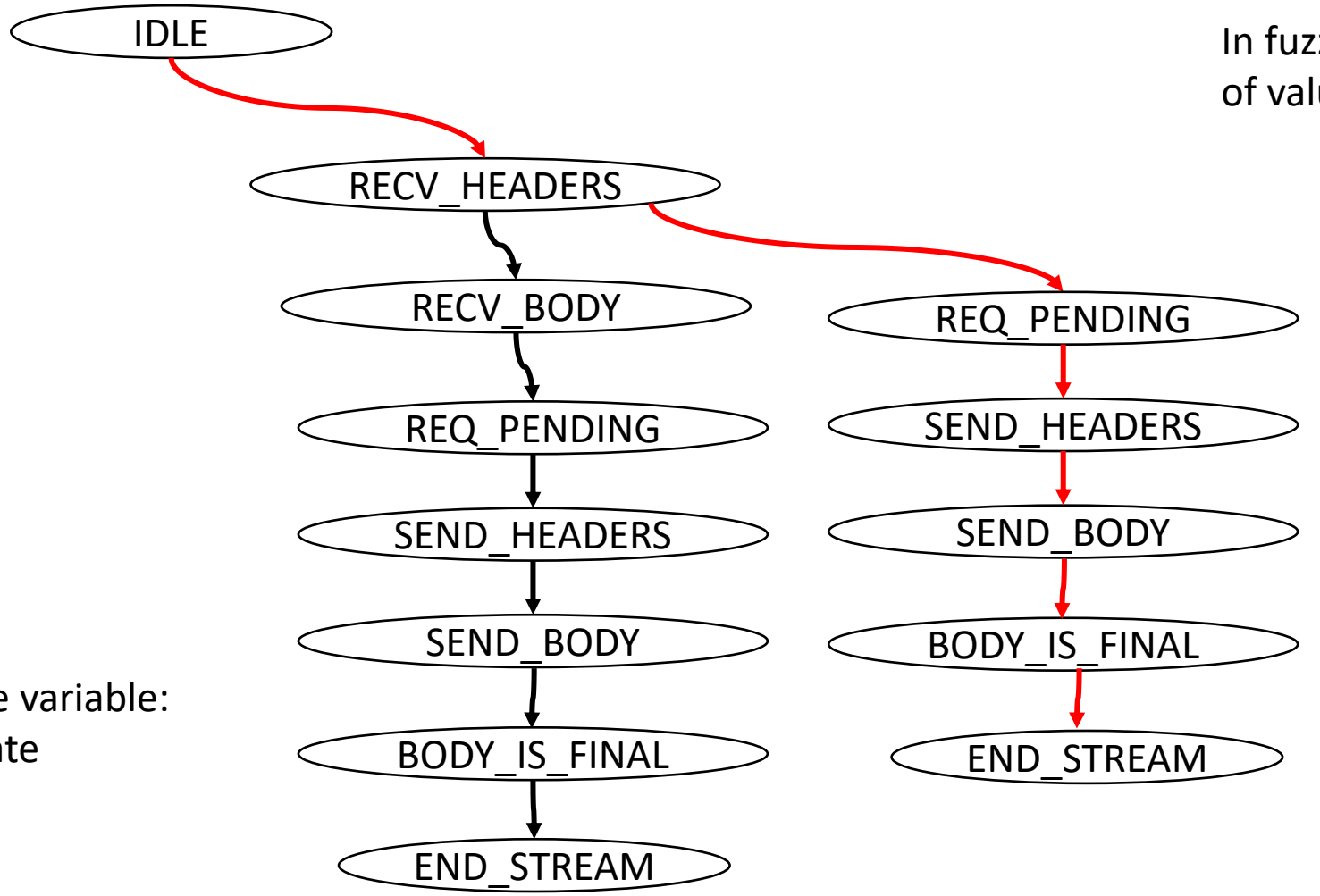
State Transition Tree Construction



In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable:
stream->state

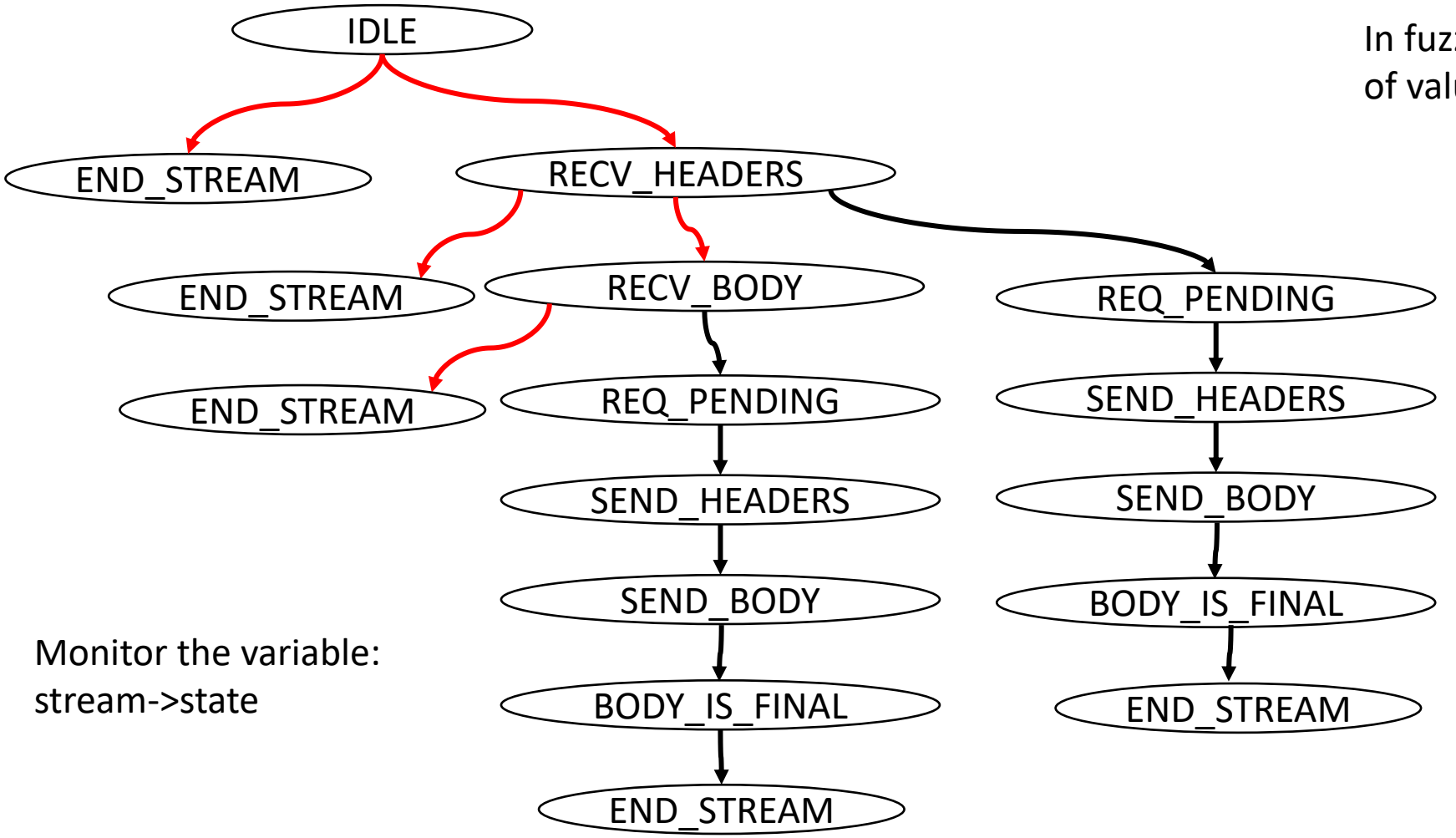
State Transition Tree Construction



In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable:
stream->state

State Transition Tree Construction

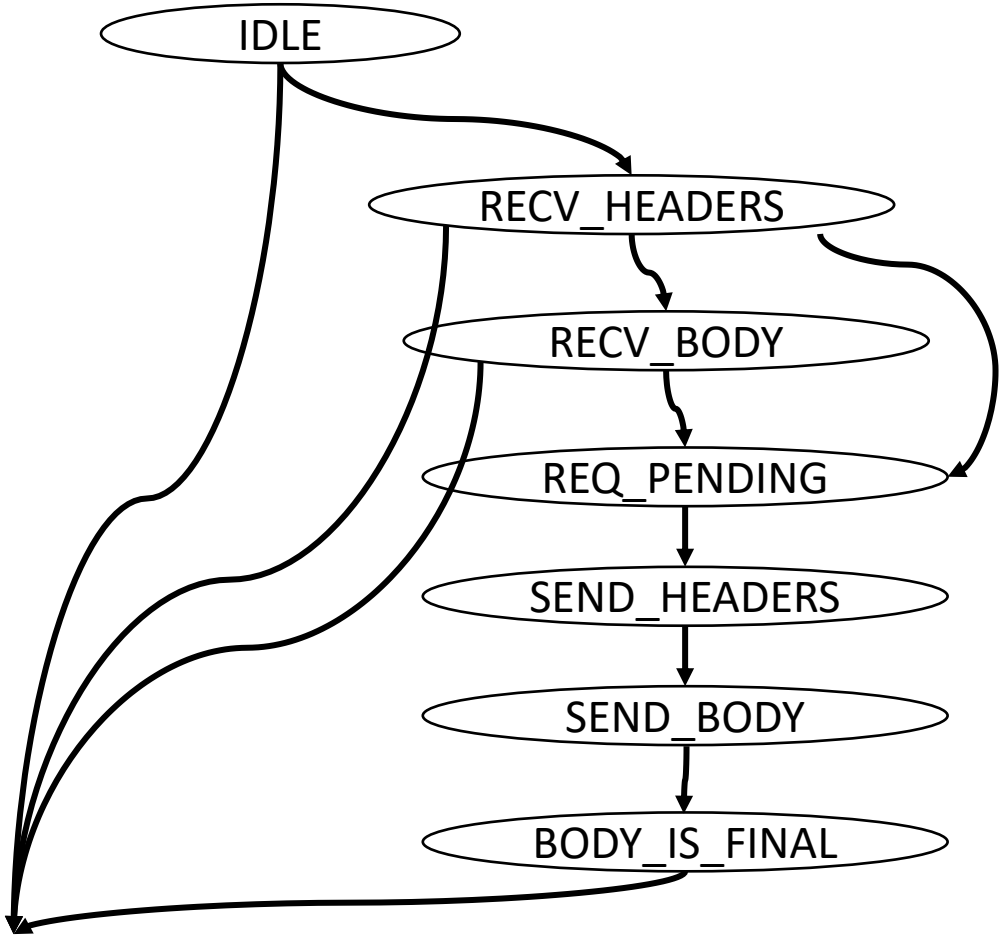


In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable:
stream->state

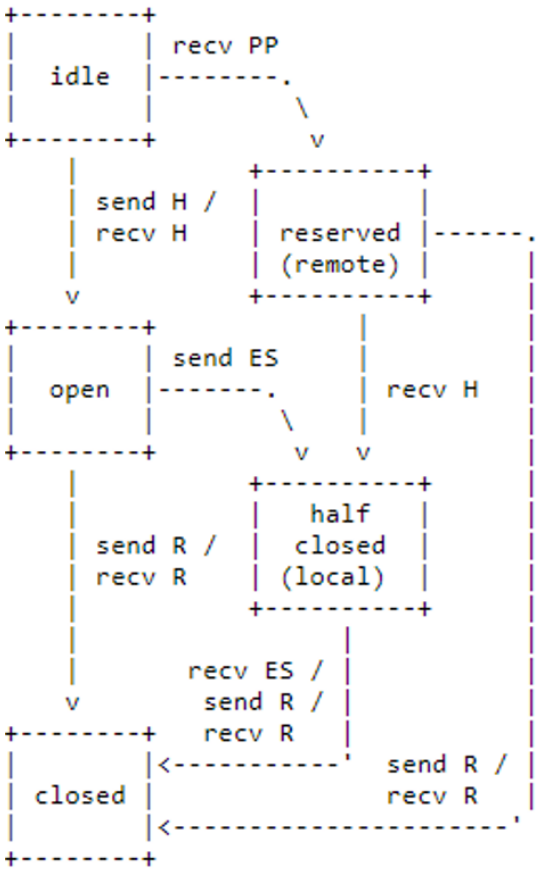
Is State Transition Tree Meaningful?

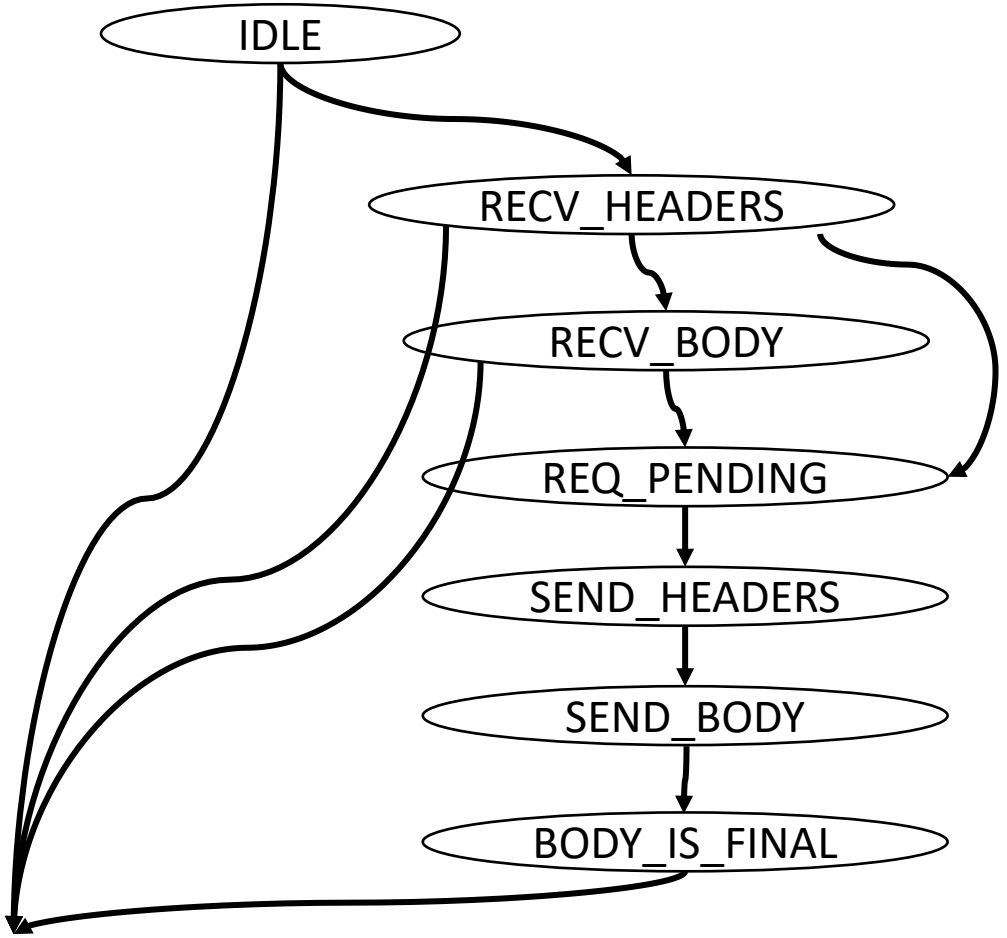
Extracting the state machine from the State Transition Tree by merging the nodes with the same value.



Is State Transition Tree Meaningful?

Compare with the official document.

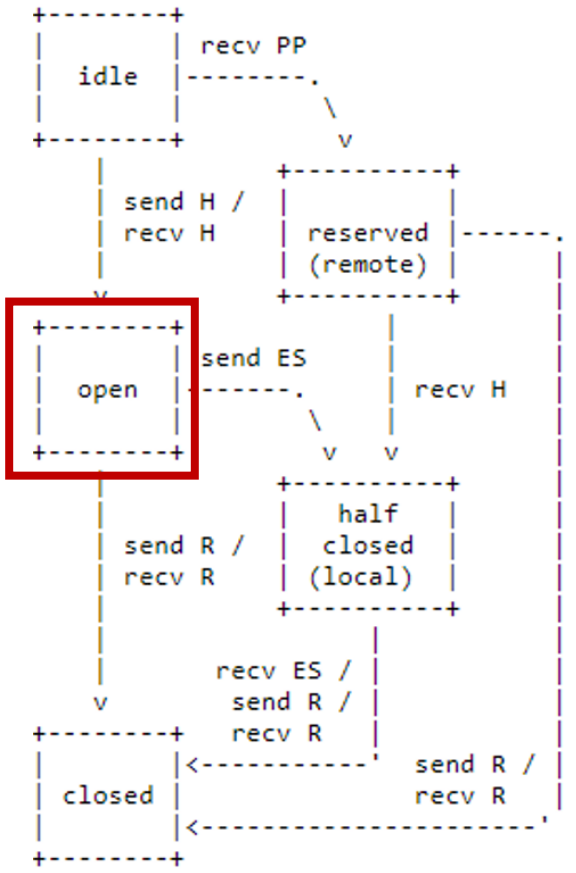


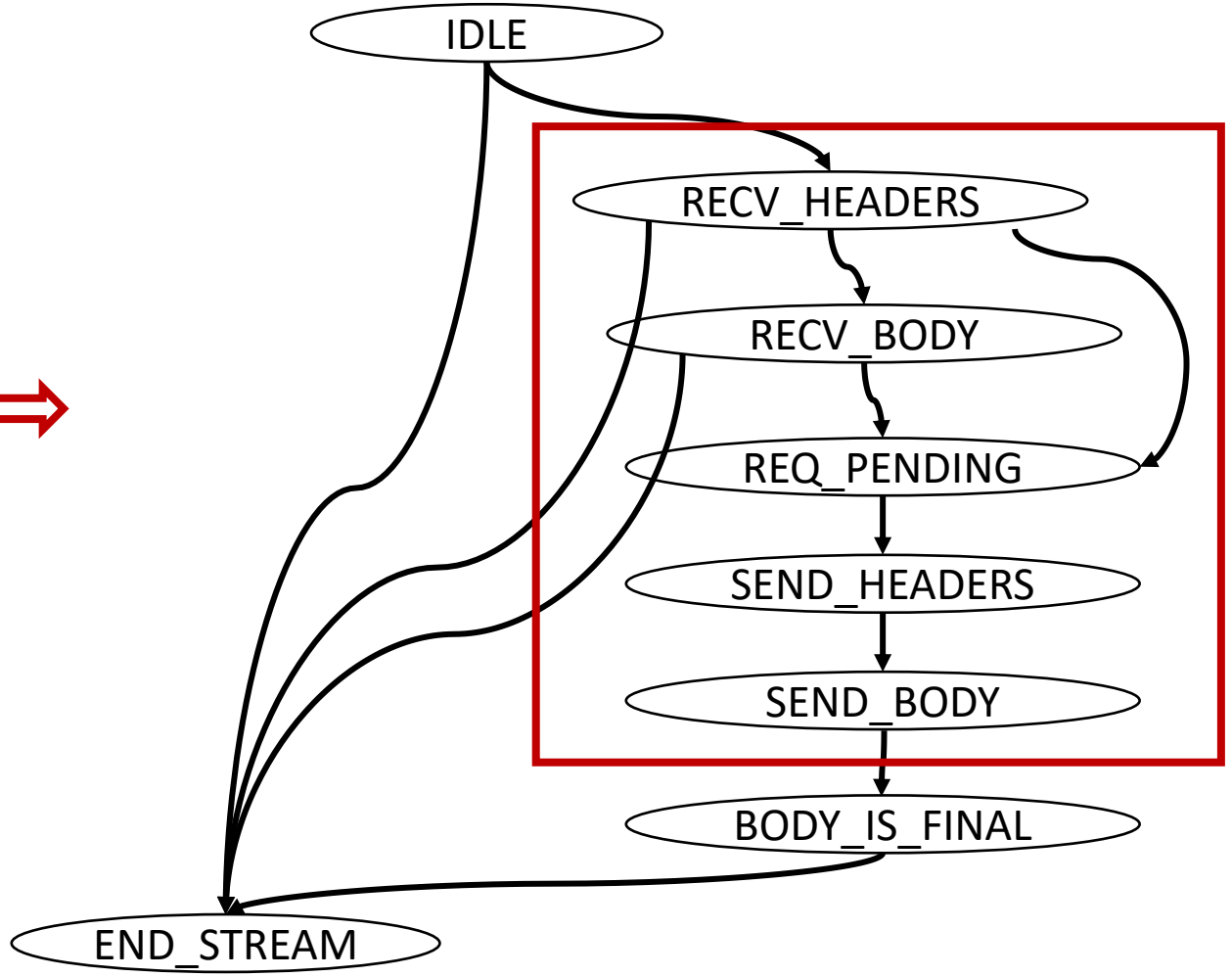


https://datatracker.ietf.org/doc/html/rfc7540#section-5.1

Is State Transition Tree Meaningful?

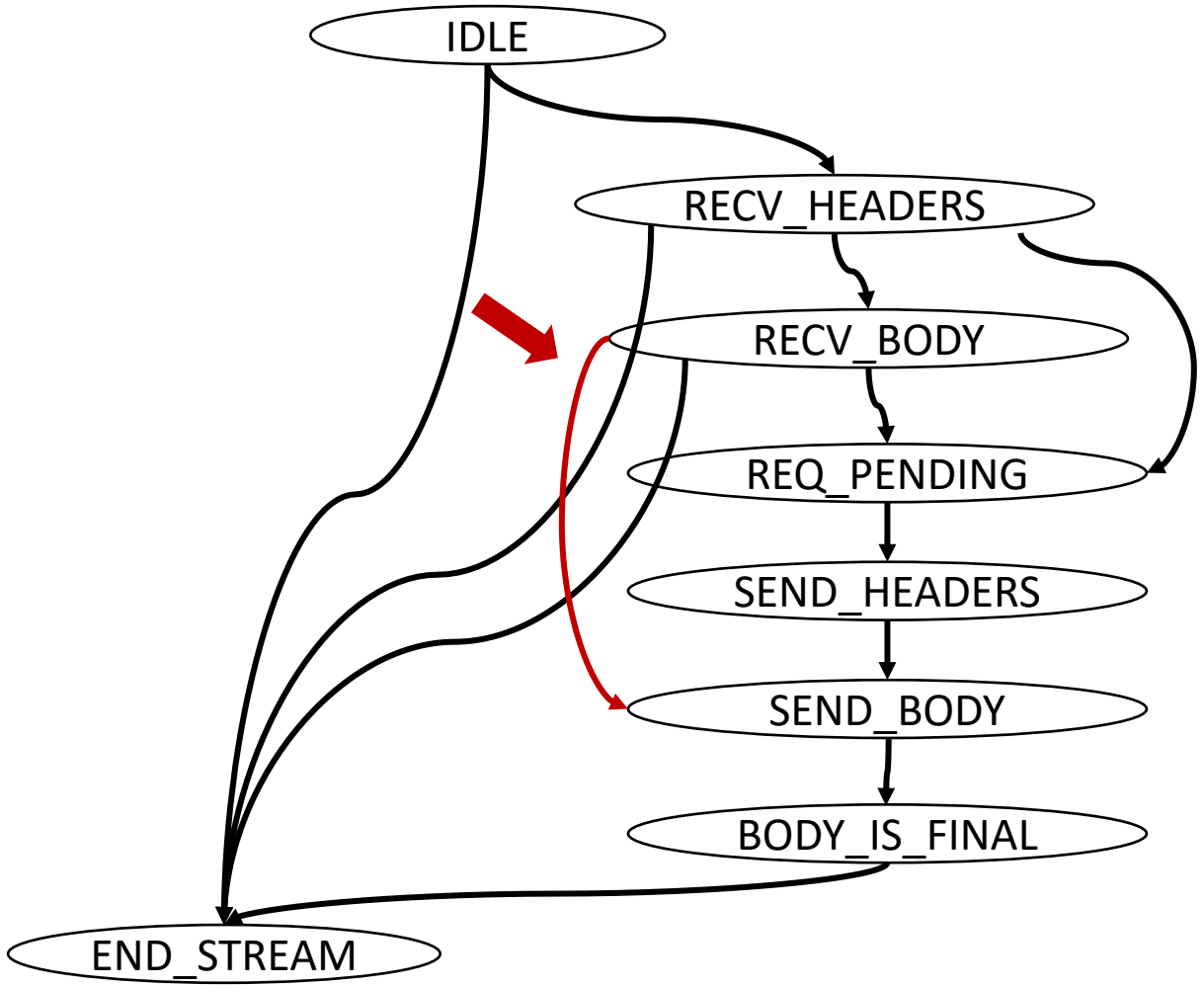
Compare with the official document.





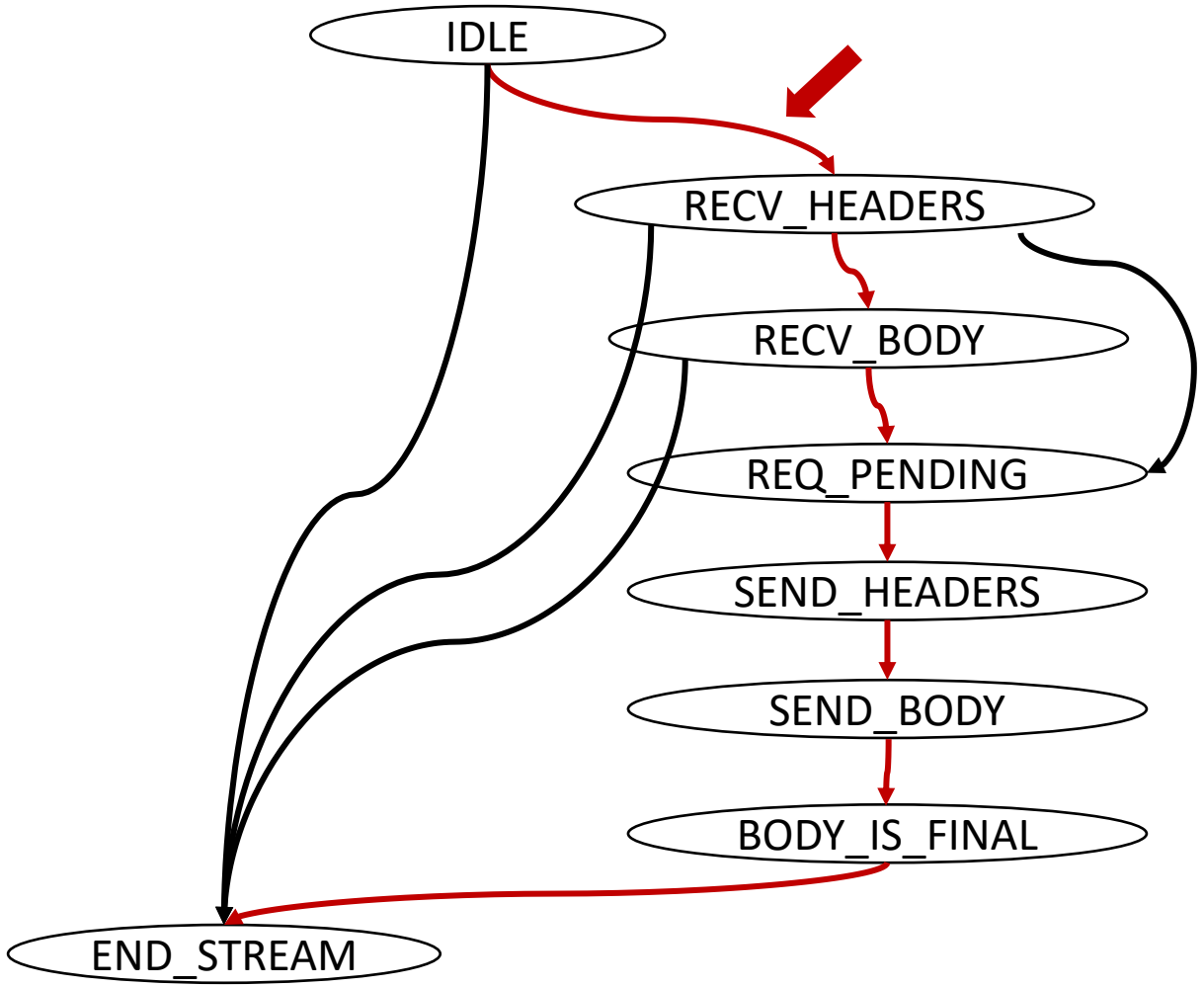
https://datatracker.ietf.org/doc/html/rfc7540#section-5.1

State Fuzzing Algorithm



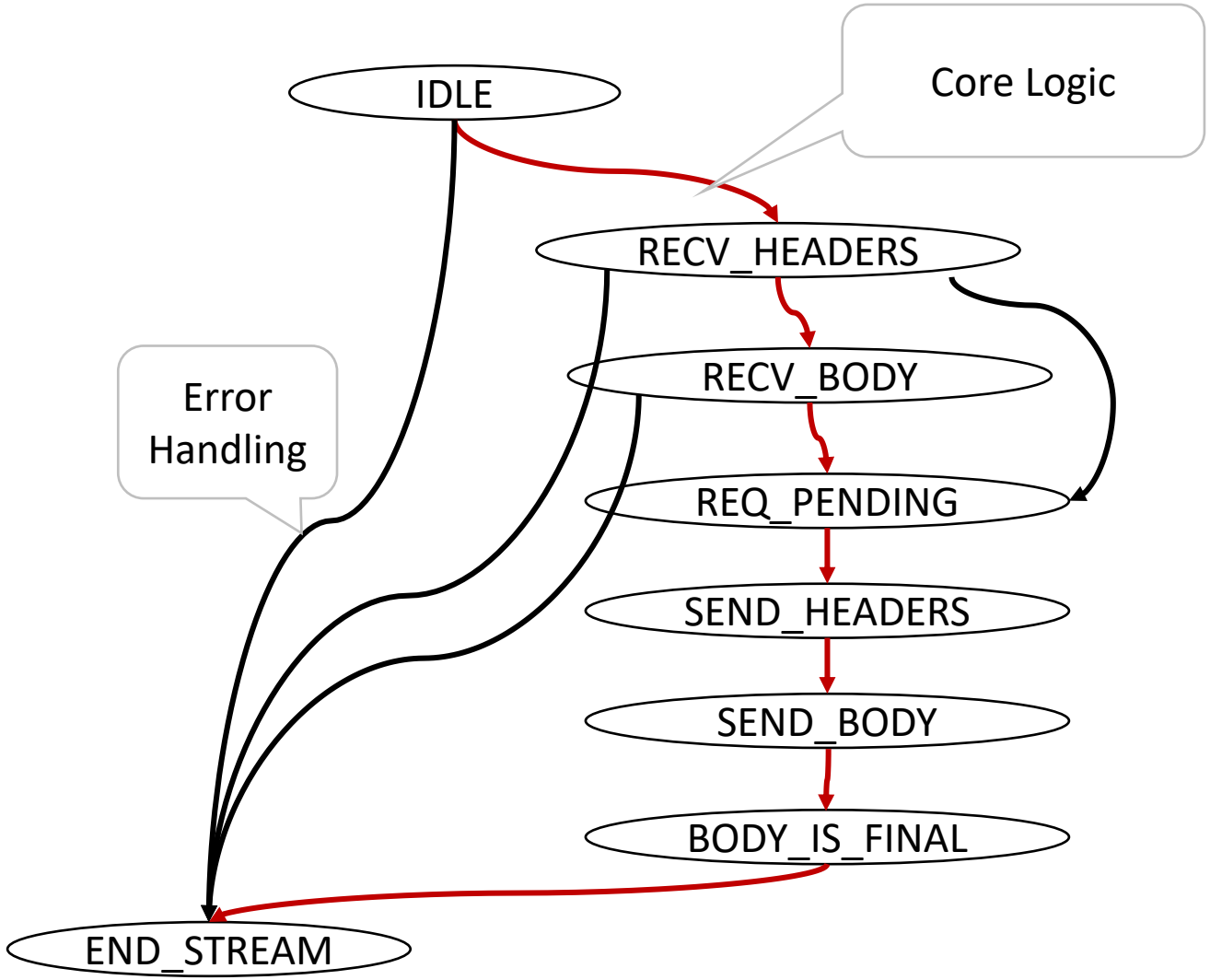
1. Save the inputs that trigger new state transitions.

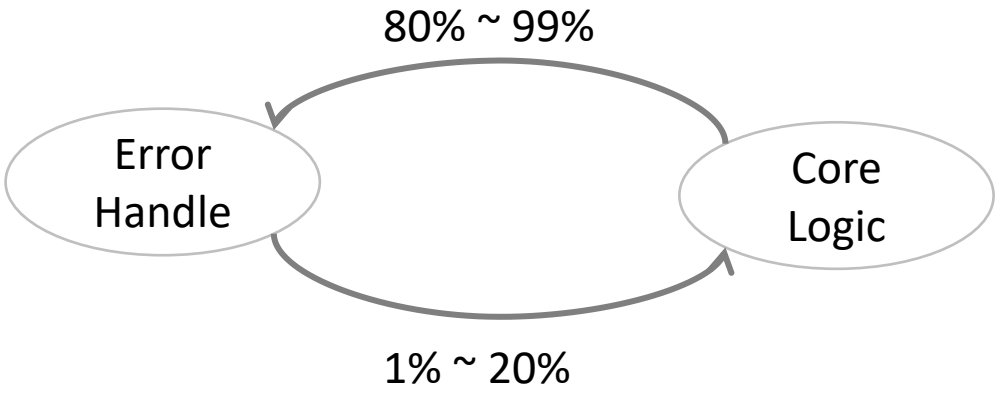
State Fuzzing Algorithm



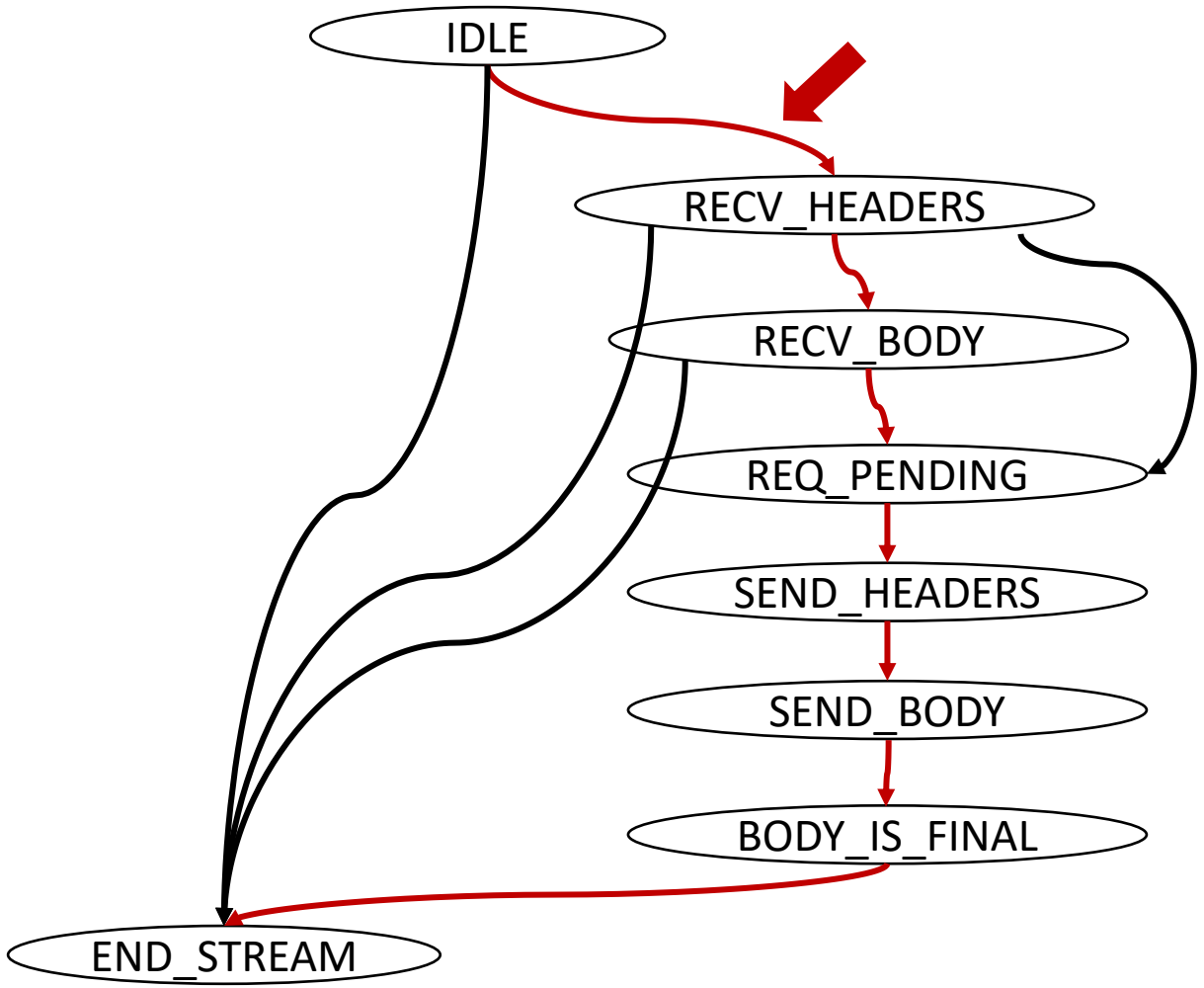
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.

Valuable State Transition Sequence





State Fuzzing Algorithm

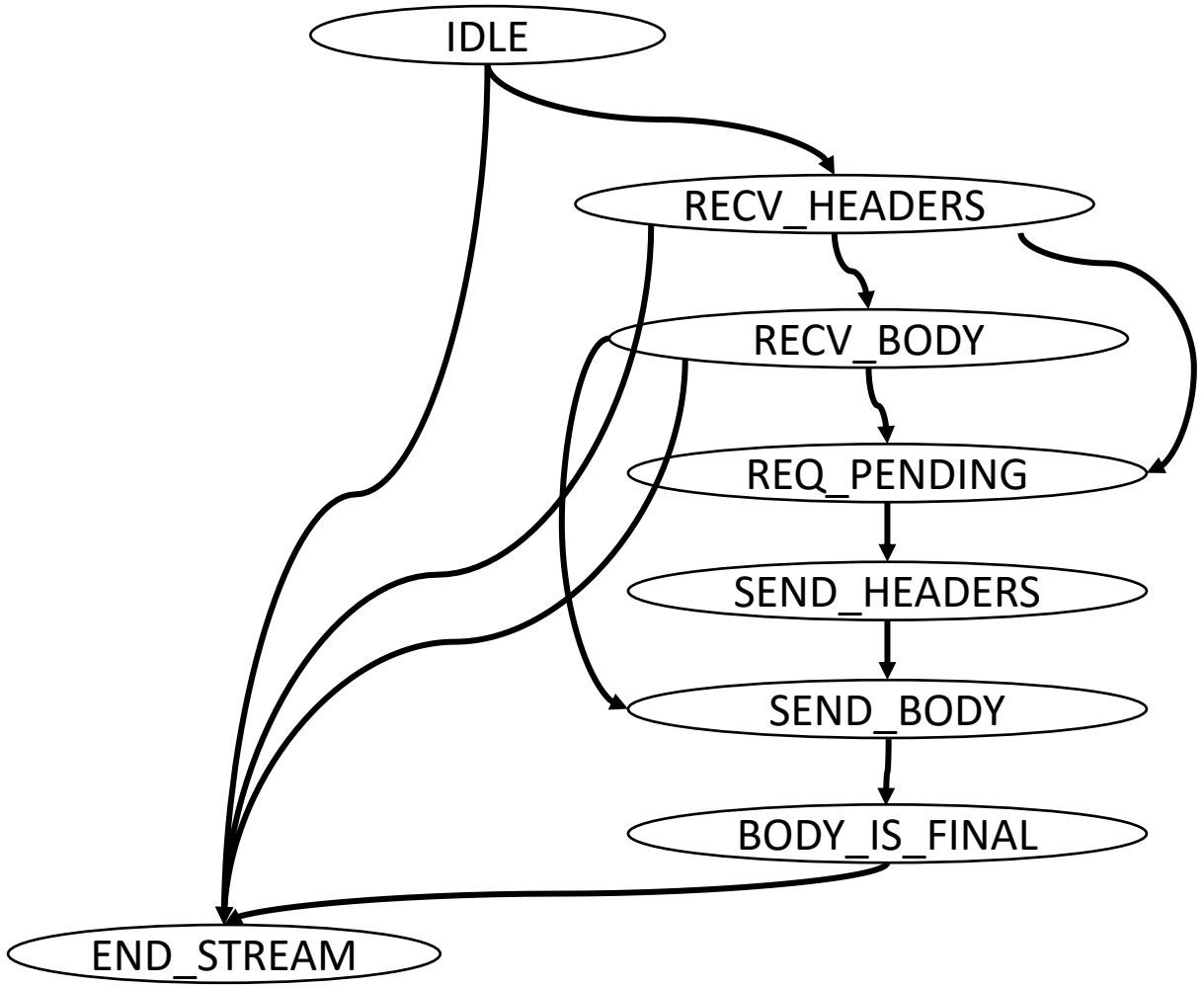


1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.

For each input I:

$$\frac{Num\ of\ inputs\ mutated\ from\ I}{Num\ of\ mutated\ inputs\ that\ still\ execute\ the\ same\ state\ sequence}$$

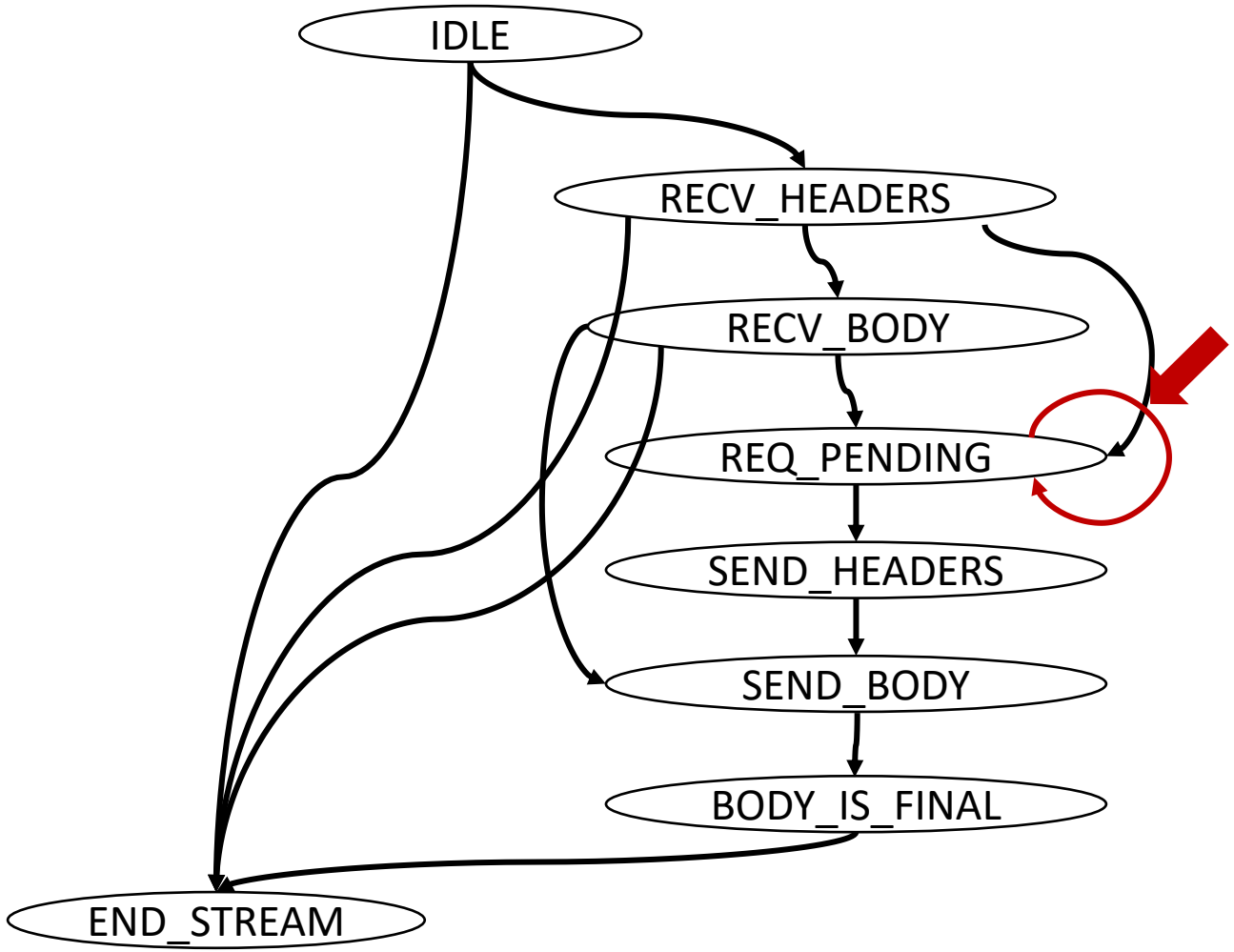
State Fuzzing Algorithm



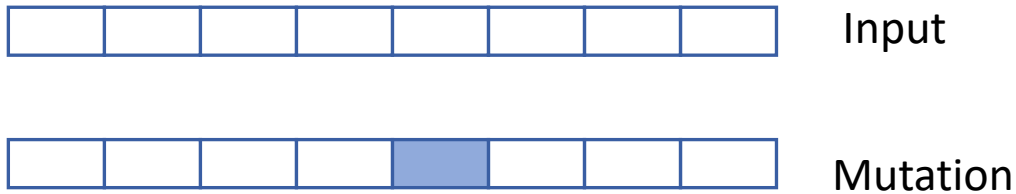
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.

Input

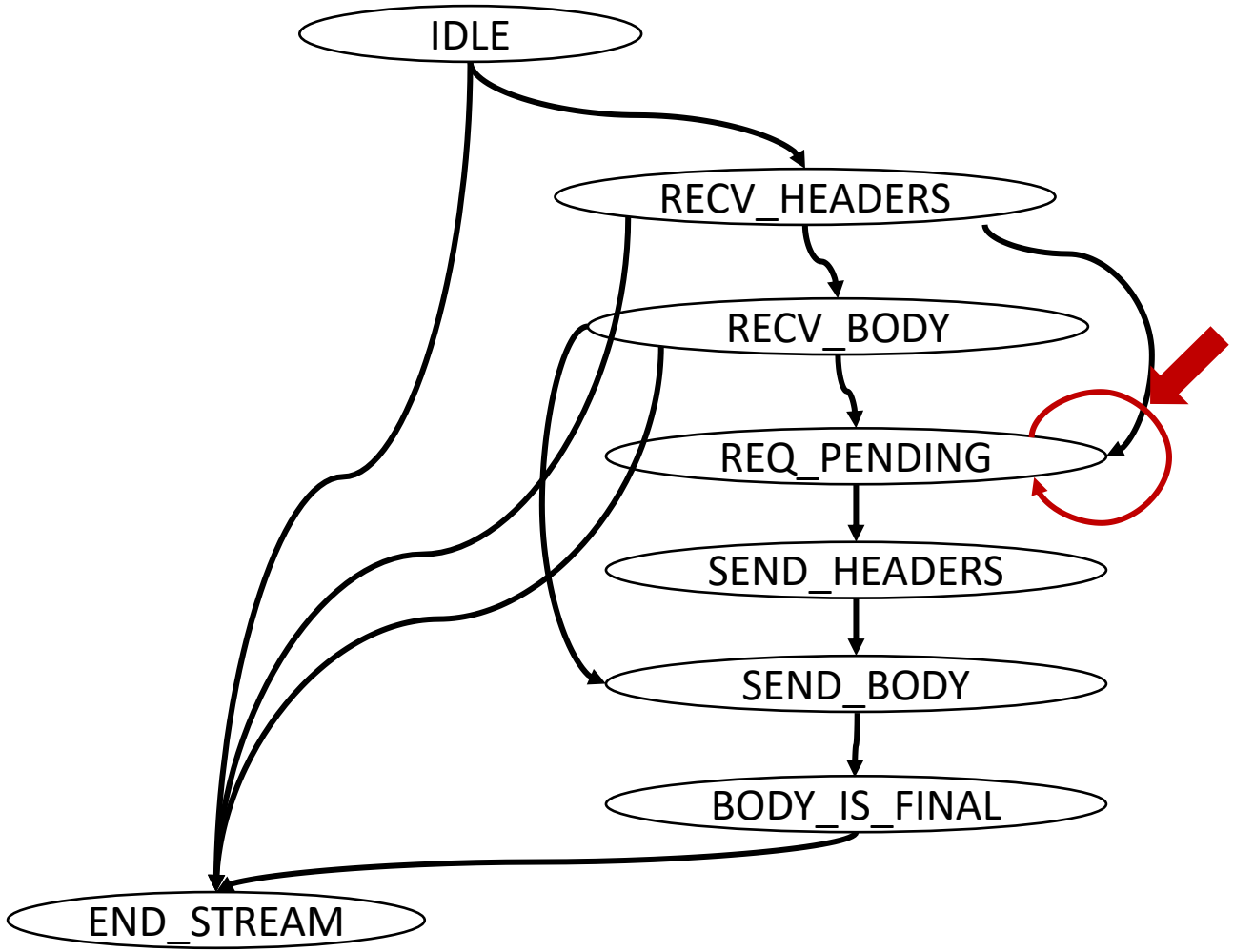
State Fuzzing Algorithm



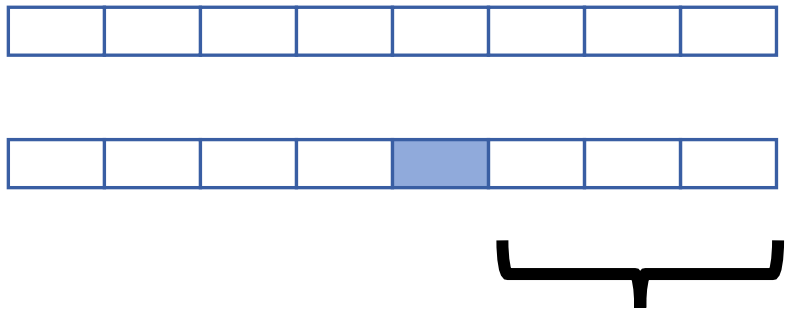
1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.



State Fuzzing Algorithm



1. Save the inputs that trigger new state transitions.
2. Assign more energy on the “core-logic” state sequences.
3. Correlate input bytes and state transitions, giving more opportunities on mutating these bytes.



More effort to mutate this part

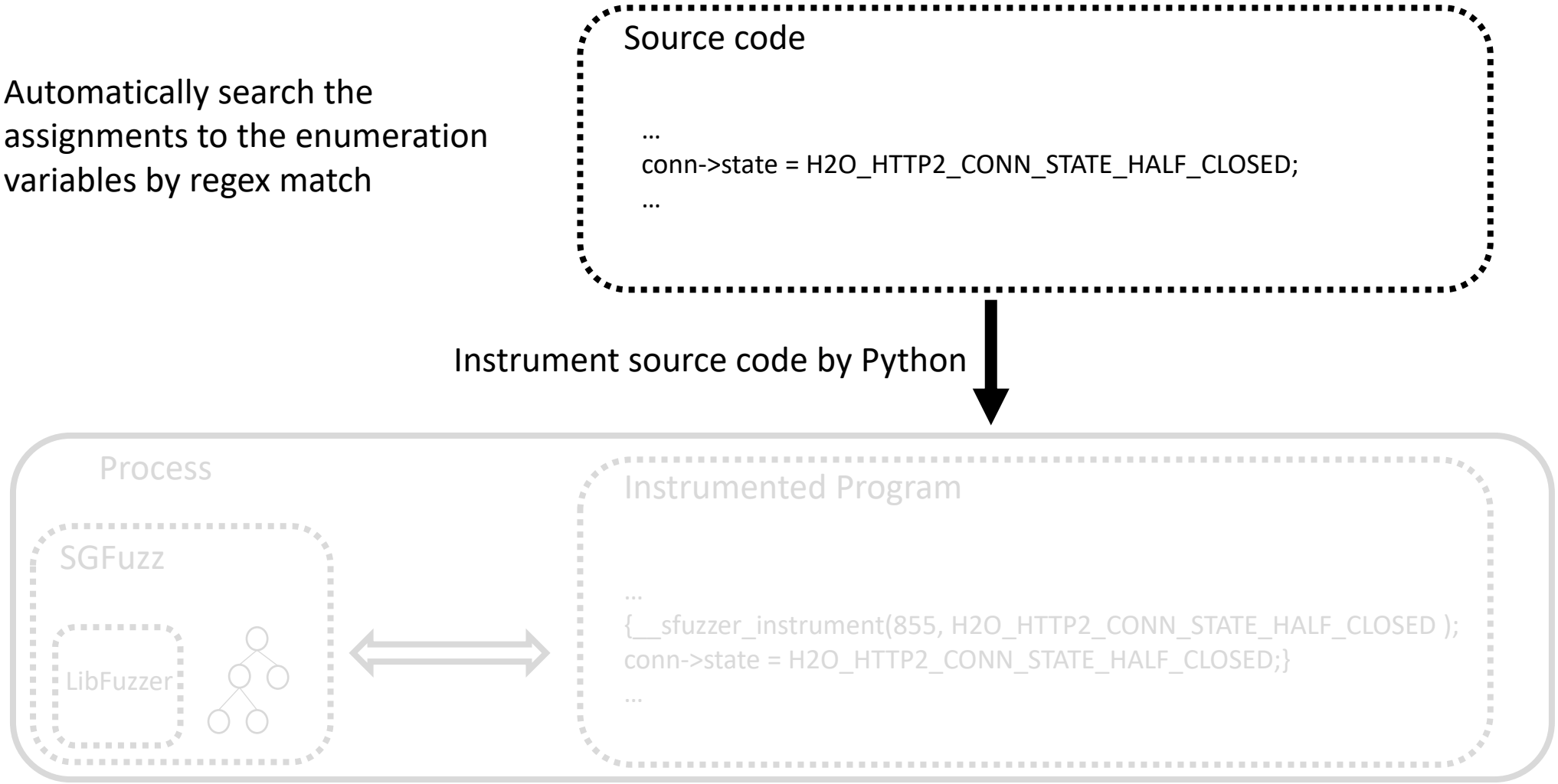
Implementation

SGFuzz is built on top of LibFuzzer:

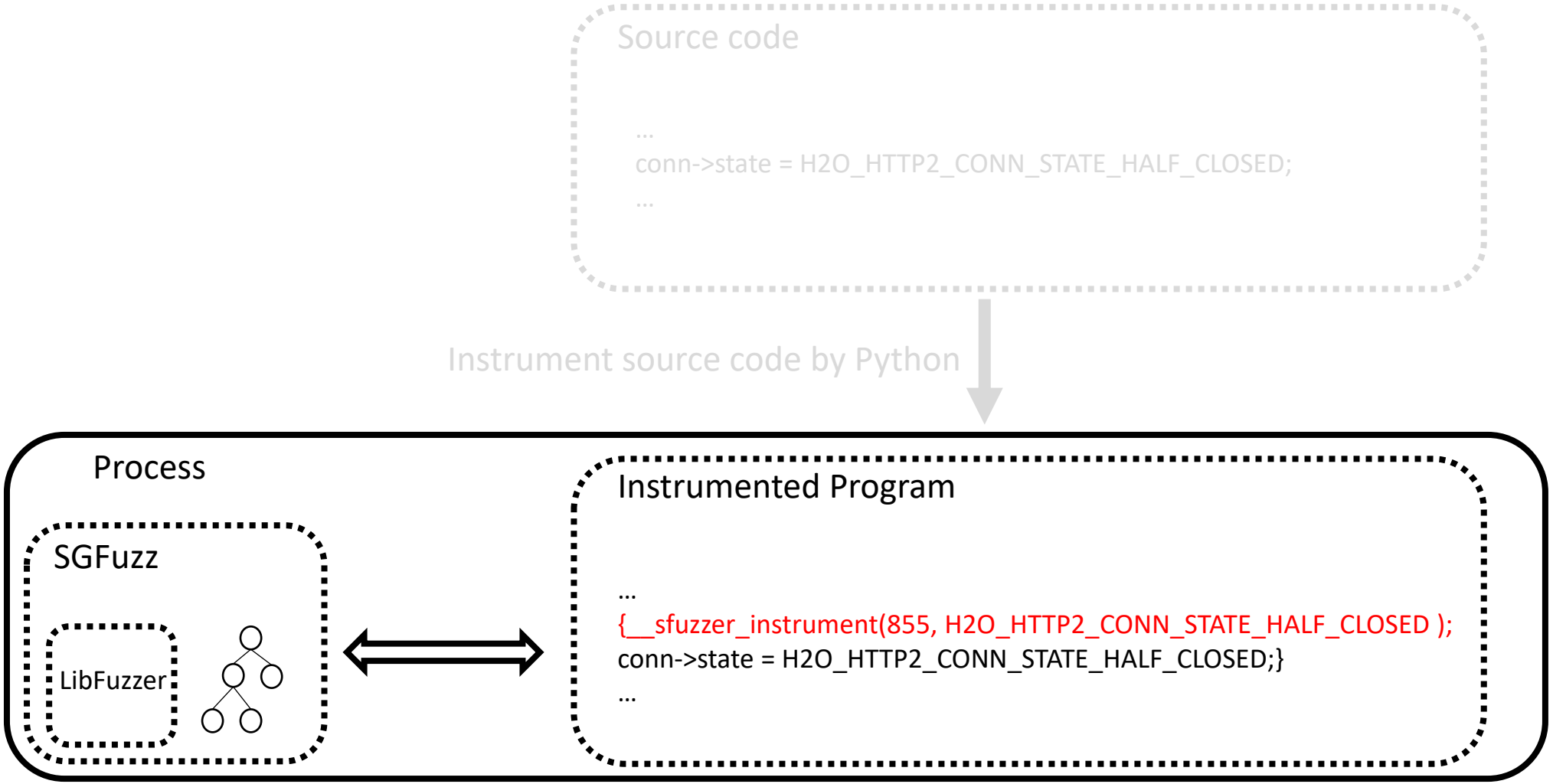
10x faster than AFL (In-memory VS Fork).

Implementation

Automatically search the assignments to the enumeration variables by regex match



Implementation



Evaluation: Benchmarks

Subject	Protocol	Fuzz Driver	Commit	Size	Share
H2O	HTTP	h2o-fuzzer-http2	1e7344	337kLoC	12kIPs
MbedTLS	SSL/TLS	dtlsserver	e483a7	138kLoC	8mIPs
Curl	Several	curl_fuzzer	aab3a7	202kLoC	-
Gstreamer	Custom	gst-discoverer	44bdad	235kLoC	15kIPs
OpenSSL	SSL/TLS	netdriver	1e08f3	673kLoC	>10mIPs
Live555	RTSP	netdriver	21.Aug'08	17kLoC	12kIPs
OwnTone	DAAP	netdriver	774d7c	37kLoC	10kIPs
DCMTK	DICOM	netdriver	24ebf4	38kLoC	3kIPs

23 hours & 20 runs

Share represents the number of IPs running it on the Internet according to Shodan(www.shodan.io)

State Transition Coverage

- We measure the number of state transition sequences in the State Transition Tree

Subject	AFLNet	LibFuzzer	IJON	SGFuzz	Factor
H2O	-	70.80	91.85	1849.30	26.1
MbedTLS	-	22.80	32.45	50.80	2.2
Curl	-	150.25	375.75	14630.80	97.3
Gstreamer	-	49.40	134.20	4067.30	82.3
OpenSSL	13.25	23.95	29.60	33.10	1.4
Live555	138.27	184.15	405.3	1162.30	6.3
OwnTone	1.00	46.40	426.00	930.15	20.0
DCMTK	68.10	189.25	267.50	6737.05	35.6

Avg: 33.9x

On average, SGFuzz covers state transition sequences 30 times more than the baseline LibFuzzer.

State Identification Effectiveness

Subject	State Transition Tree		
	All Nodes	State	Percentage
H2O	6418	6417	99.98%
MbedTLS	167	167	100.00%
Curl	35690	35629	99.83%
Gstreamer	11240	11224	99.86%
OpenSSL	817	789	96.57%
Live555	17446	17446	100.00%
OwnTone	3671	3671	100.00%
DCMTK	27178	27109	99.75%
			Avg: 99.50%

In our data structure State Transition Tree, 99.5% nodes are related to the true states.

New Bugs

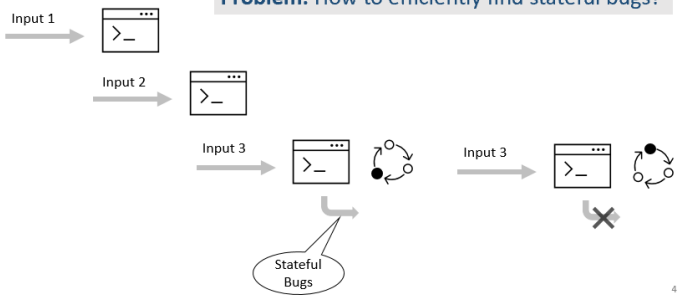
Subject	Version	Type	Stateful	CVE
Live555	1.08	Stack-based overflow in liveMedia/MP3FileSource.cpp	✓	CVE-2021-38380
Live555	1.08	Heap use after free in liveMedia/MatroskaFile.cpp	✓	CVE-2021-38381
Live555	1.08	Heap use after free in liveMedia/MPEG1or2Demux.cpp	✓	CVE-2021-38382
Live555	1.08	Memory leak in liveMedia/AC3AudioStreamFramer.cpp	✓	CVE-2021-39282
Live555	1.08	Assertion in UsageEnvironment/UsageEnvironment.cpp	✓	CVE-2021-39283
Live555	1.08	Heap-based overflow in BasicUsageEnvironment/BasicTaskScheduler.cpp	✓	CVE-2021-41396
Live555	1.08	Memory leak in liveMedia/MPEG1or2Demux.cpp	✓	CVE-2021-41397
OwnTone	28.2	Heap use after free in src/misc.c	✗	CVE-2021-38383
DCMTK	3.6.6	Memory leak in dcmnet/libsrc/dulparse.cc	✗	CVE-2021-41687
DCMTK	3.6.6	Memory leak in dcmnet/libsrc/dulparse.cc	✓	CVE-2021-41688
DCMTK	3.6.6	Heap use after free in dcmqrdb/libsrc/dcmqrsrv.cc	✓	CVE-2021-41689
DCMTK	3.6.6	Heap-based overflow in dcmnet/libsrc/diutil.cc	✓	CVE-2021-41690

We found 12 previously unknown bugs in 23 hours, and 10 of 12 are stateful bugs.

Conclusion

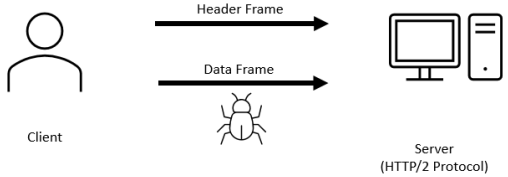
Challenge: Bugs in Stateful Programs

Problem: How to efficiently find stateful bugs?



4

Protocol Implementations (Stateful Programs)



8

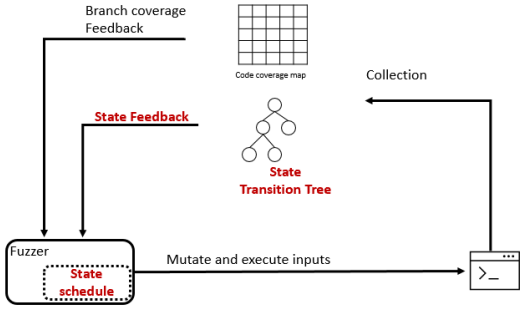
Insight

Approximating state variables by the variables with named constants.

- There may be variables take named constants that are not state variables (e.g., configuration variables, error code variables)
- In our evaluation, over 99% of extracted variables are true states.

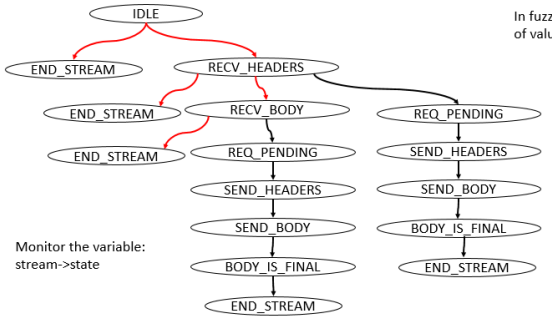
15

Stateful Greybox Fuzzing



16

State Transition Tree Construction



In fuzzing, we monitor the changes of values of enumeration variables.

Monitor the variable: stream->state

20

New Bugs

Subject	Version	Type	Stateful	CVE
Live555	1.08	Stack-based overflow in liveMedia/MP3FileSource.cpp	✓	CVE-2021-38380
Live555	1.08	Heap use after free in liveMedia/MatroskaFile.cpp	✓	CVE-2021-38381
Live555	1.08	Heap use after free in liveMedia/MPEG1or2Demux.cpp	✓	CVE-2021-38382
Live555	1.08	Memory leak in liveMedia/AC3AudioStreamFramer.cpp	✓	CVE-2021-39282
Live555	1.08	Assertion in UsageEnvironment/UsageEnvironment.cpp	✓	CVE-2021-39283
Live555	1.08	Heap-based overflow in BasicUsageEnvironment/BasicTaskScheduler.cpp	✓	CVE-2021-41396
Live555	1.08	Memory leak in liveMedia/MPEG1or2Demux.cpp	✓	CVE-2021-41397
OwnTone	28.2	Heap use after free in src/misc.c	✗	CVE-2021-38383
DCMTK	3.6.6	Memory leak in dcmnet/libsrc/dulparse.cc	✗	CVE-2021-41687
DCMTK	3.6.6	Memory leak in dcmnet/libsrc/dulparse.cc	✓	CVE-2021-41688
DCMTK	3.6.6	Heap use after free in dcmqrdb/libsrc/dcmqrsrv.cc	✓	CVE-2021-41689
DCMTK	3.6.6	Heap-based overflow in dcmnet/libsrc/diutil.cc	✓	CVE-2021-41690

We found 12 previously unknown bugs in 23 hours, and 10 of 12 are stateful bugs.

37

jinsheng@comp.nus.edu.sg
https://github.com/bajinsheng/SGFuzz