Hands-on symbolic search & test generation with Apalache

igor@konnov.phd

NVidia FM Week — Online, Nov 20, 2025

Igor Konnov

Independent security and formal methods researcher 🇦🇹 Vienna, Austria — 2024-2025

- Verifying consensus protocols and smart contracts
- Improving Apalache, working on new tools

Principal research scientist at Informal Systems (Web3 – blockchains) — 2019-2023

- Leading development of Apalache and Quint

Postdoc, assistant professor & permanent researcher at TU Wien & Inria — 2011-2019

🇦🇹 🇫🇷

This talk

1. A bit of history ☕
2. Symbolic search with Apalache 😵
3. TFTP Protocol in TLA$^+$ ⚙️🔩
4. Testing loop for tftpd-hpa with Apalache and Claude 🦾

[github.com/apalache-mc/apalache]

[apalache-mc.org]

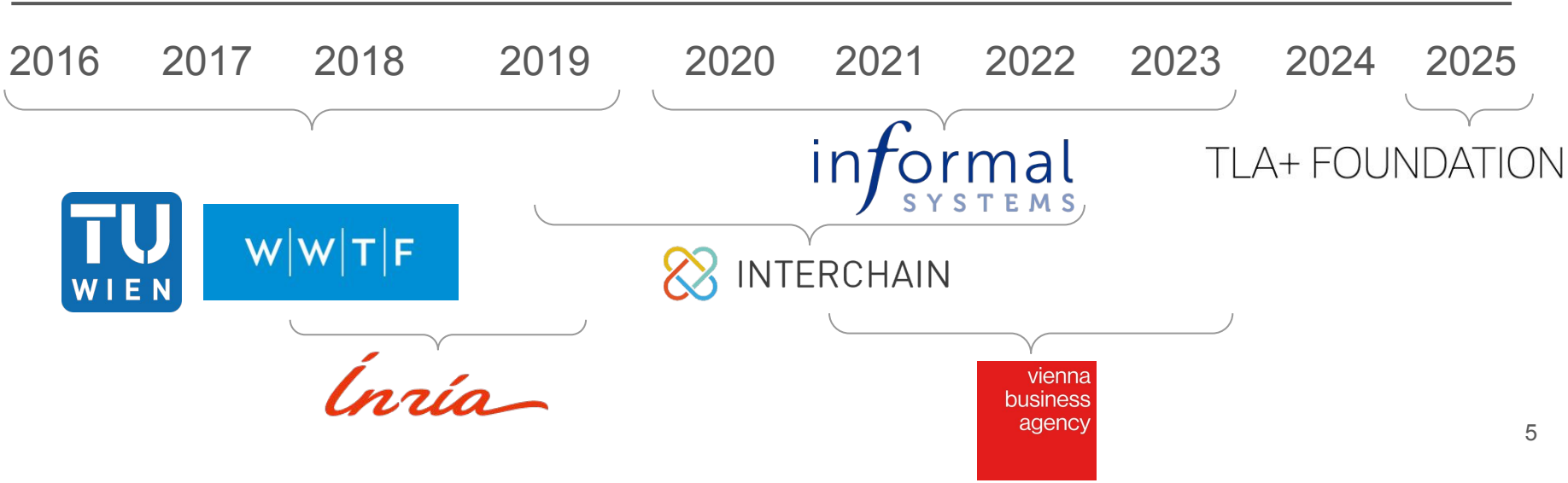
APALACHE

A symbolic model checker for TLA+

build passing

Apalache translates TLA+ into the logic supported by SMT solvers such as Microsoft Z3. Apalache can check inductive invariants (for fixed or bounded parameters) and check safety of bounded executions (bounded model checking). To see the list of supported TLA+ constructs, check the supported features. In general, Apalache runs under the same assumptions as TLC. However, Apalache benefits from constraint solving and can handle potentially larger state-spaces, e.g., involving integer clocks and Byzantine faults.

To learn more about TLA+, visit Leslie Lamport's page on TLA+ and his Video course.


2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
informal SYSTEMS
TLA+ FOUNDATION
TU WIEN
WWTF
INTERCHAIN
Inria
vienna business agency

Epoch 1

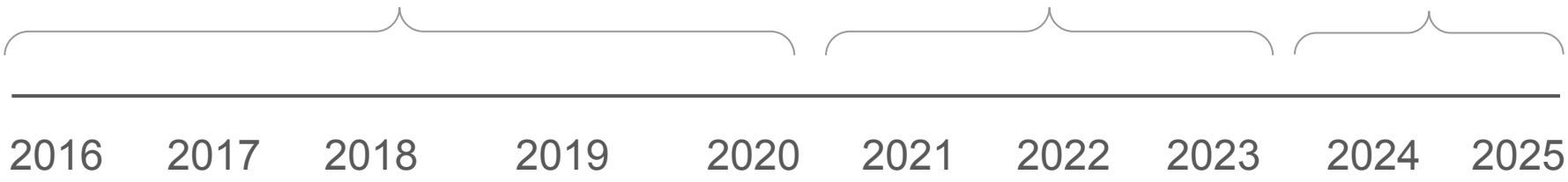
1. SMT transpiler
2. Type checker
3. Bounded model checker

Epoch 2

1. Random. symbolic execution
2. Data generators

Epoch 3

1. Parallelization
2. DIY search



what academia need

what engineers need

what customers need

WWTF

Thanh-Hai Tran

Marijana Lazić

Josef Widder

TU WIEN

vienna business agency

Jure Kukovec

Shon Feder

Gabriela Moreira
@bugarela

Igor Konnov
@konnov

Thomas Pani

Andrey Kupriyanov

Philip Offtermatt

Rodrigo Otoni

informal SYSTEMS

INTERCHAIN

Inria

Università della Svizzera italiana

Matter Labs

ZKsync

Governance Protocol (2024)

ChonkyBFT consensus (2024-2025)

L2 Governance Protocol (2025)

Accountability in Ethereum 3-slot finality (2024)

TetraBFT consensus (2024)

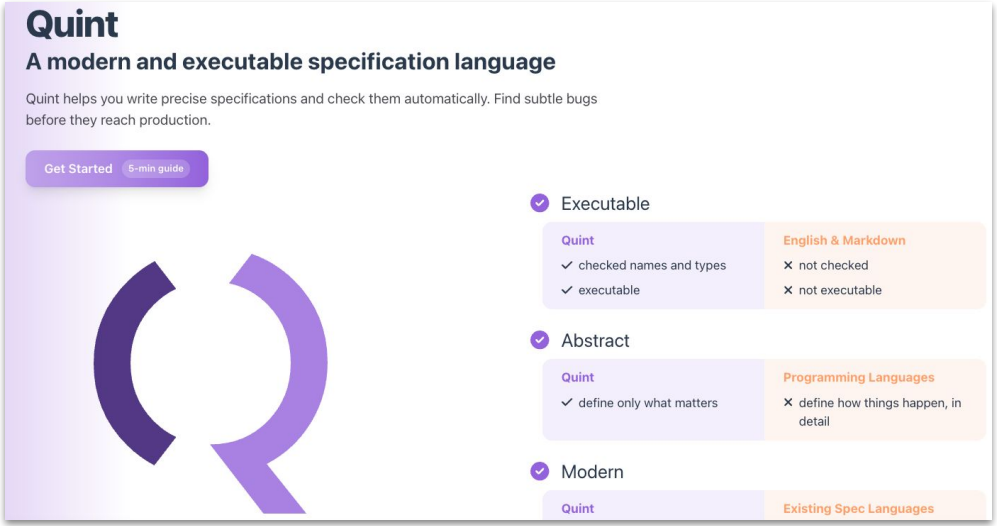
Tendermint BFT consensus (2020)

Ben-Or's consensus (2024)

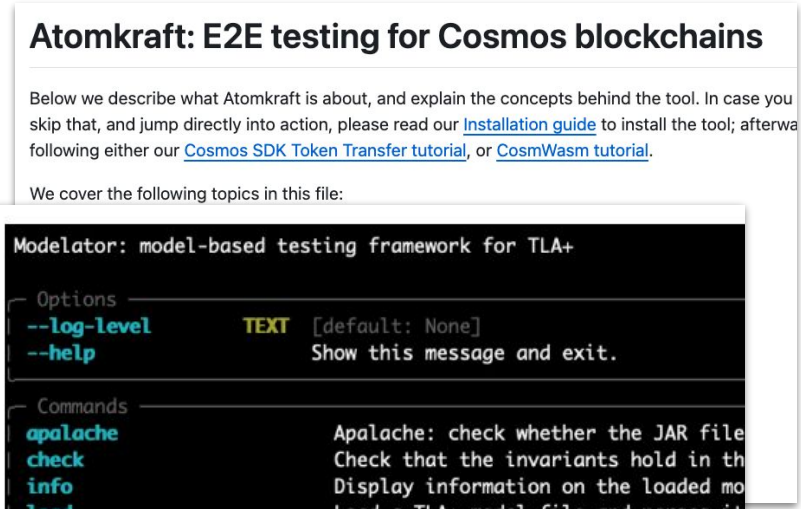
Tendermint light client (2020)

Offshoot projects

[quint-lang.org]



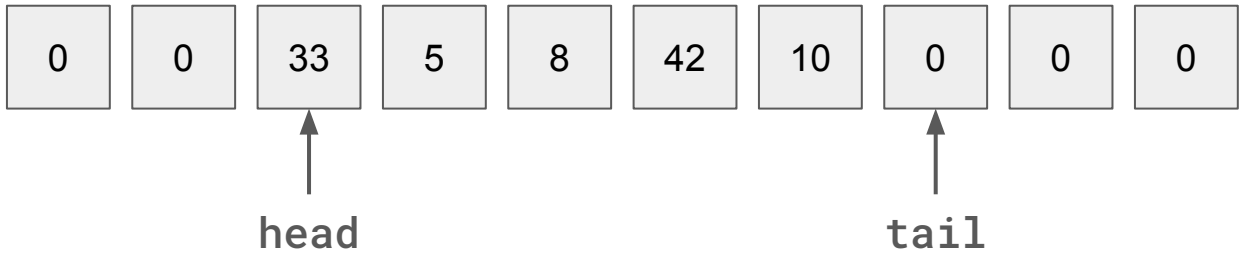
[github.com/informalsystems/atomkraft]



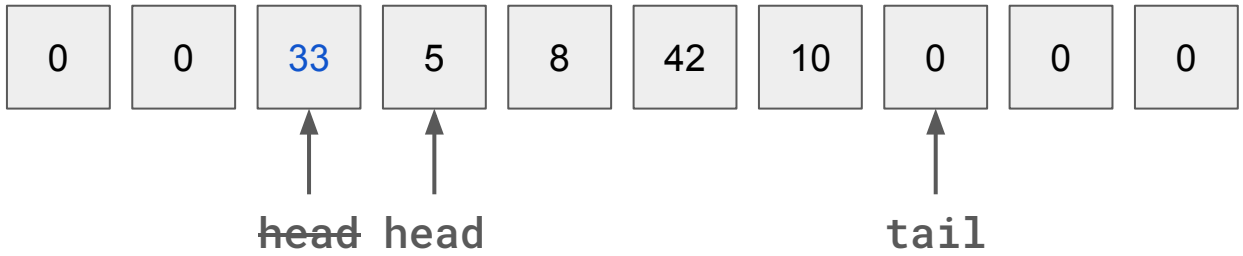
informal SYSTEMS since 2021

Symbolic search with Apalache

Example: Circular buffer

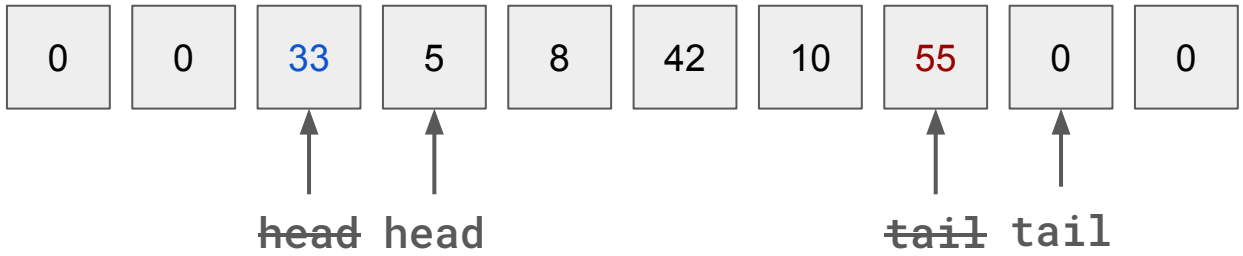


Example: Circular buffer



1. GET → 33, head’ = (head + 1) % N

Example: Circular buffer



1. GET → 33, head’ = (head + 1) % N
2. PUT(55), tail’ = (tail + 1) % N

```
CONSTANTS
    \* Size of the circular buffer.
    \* @type: Int;
    BUFFER_SIZE,
    \* The set of possible buffer elements.
    \* @type: Set(Int);
    BUFFER_ELEMS

ASSUME BUFFER_SIZE > 0

VARIABLES
    \* The integer buffer of size BUFFER_SIZE.
    \* @type: Int -> Int;
    buffer,
    \* Index of the next element to POP.
    \* @type: Int;
    head,
    \* Index of the next free slot for PUSH.
    \* @type: Int;
    tail,
    \* Number of elements currently stored.
    \* @type: Int;
    count
```

```
Put(x) ==
  Put::
  LET nextTail == (tail + 1) % BUFFER_SIZE IN
  /\ buffer' = [buffer EXCEPT ![tail] = x]
  /\ head' = head
  /\ tail' = nextTail
  /\ count' = count + 1


Get ==
  Get::
  LET nextHead == (head + 1) % BUFFER_SIZE IN
  /\ count > 0
  /\ UNCHANGED buffer
  /\ head' = nextHead
  /\ tail' = tail
  /\ count' = count - 1
```

```
41  Put(x) ==
42    Put::
43    LET nextTail == (tail + 1) % BUFFER_SIZE IN
44    /\ buffer' = [buffer EXCEPT ![tail] = x]
45    /\ head' = head
46    /\ tail' = nextTail
47    /\ count' = count + 1

50  Get ==
51    Get::
52    LET nextHead == (head + 1) % BUFFER_SIZE IN
53    /\ count > 0
54    /\ UNCHANGED buffer
55    /\ head' = nextHead
56    /\ tail' = tail
57    /\ count' = count - 1
```

```
33  \* Initial state
34  Init ==
35    /\ buffer = [ i \in 0..(BUFFER_SIZE - 1) |-> 0 ]
36    /\ head = 0
37    /\ tail = 0
38    /\ count = 0

59  \* Either Put or Get may happen in any step.
60  Next ==
61      \/ \E x \in BUFFER_ELEMS:
62          Put(x)
63      \/ Get
```

```
70  \* Safety property we *intend* to hold, but it is violated:
71  \* count must never exceed the buffer capacity.
72  SafeInv == count <= BUFFER_SIZE
```

```
41  Put(x) ==
42    Put::
43    LET nextTail == (tail + 1) % BUFFER_SIZE IN
44    /\ buffer' = [buffer EXCEPT ![tail] = x]
45    /\ head' = head
46    /\ tail' = nextTail
47    /\ count' = count + 1

50  Get ==
51    Get::
52    LET nextHead == (head + 1) % BUFFER_SIZE IN
53    /\ count > 0
54    /\ UNCHANGED buffer
55    /\ head' = nextHead
56    /\ tail' = tail
57    /\ count' = count - 1
```

```
33  \* Initial state
34  Init ==
35    /\ buffer = [ i \in 0..(BUFFER_SIZE - 1) |-> 0 ]
36    /\ head = 0
37    /\ tail = 0
38    /\ count = 0

59  \* Either Put or Get may happen in any step.
60  Next ==
61      \/ \E x \in BUFFER_ELEMS:
62          Put(x)
63      \/ Get
```

How do we find this violation?

```
70  \* Safety property we *intend* to hold, but it is violated:
71  \* count must never exceed the buffer capacity.
72  SafeInv == count <= BUFFER_SIZE
```

Instance `MC10u8_BuggyCircularBuffer`:

- BUFFER_SIZE = 10
- BUFFER_ELEMS = 0..255

Explicit-state model checker TLC:

over 40 min on 30 CPU cores

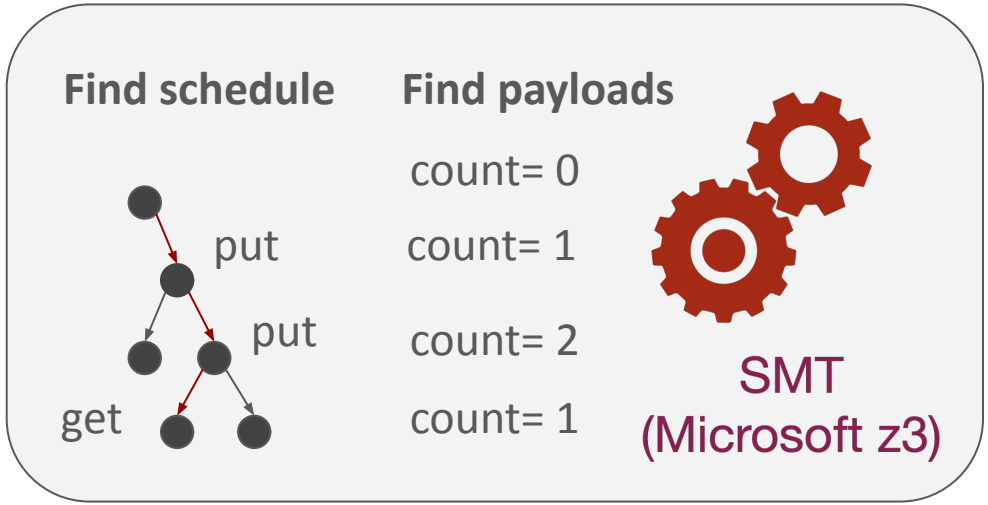
over 3 billion states

400G disk space ⇒ “No space left on device”

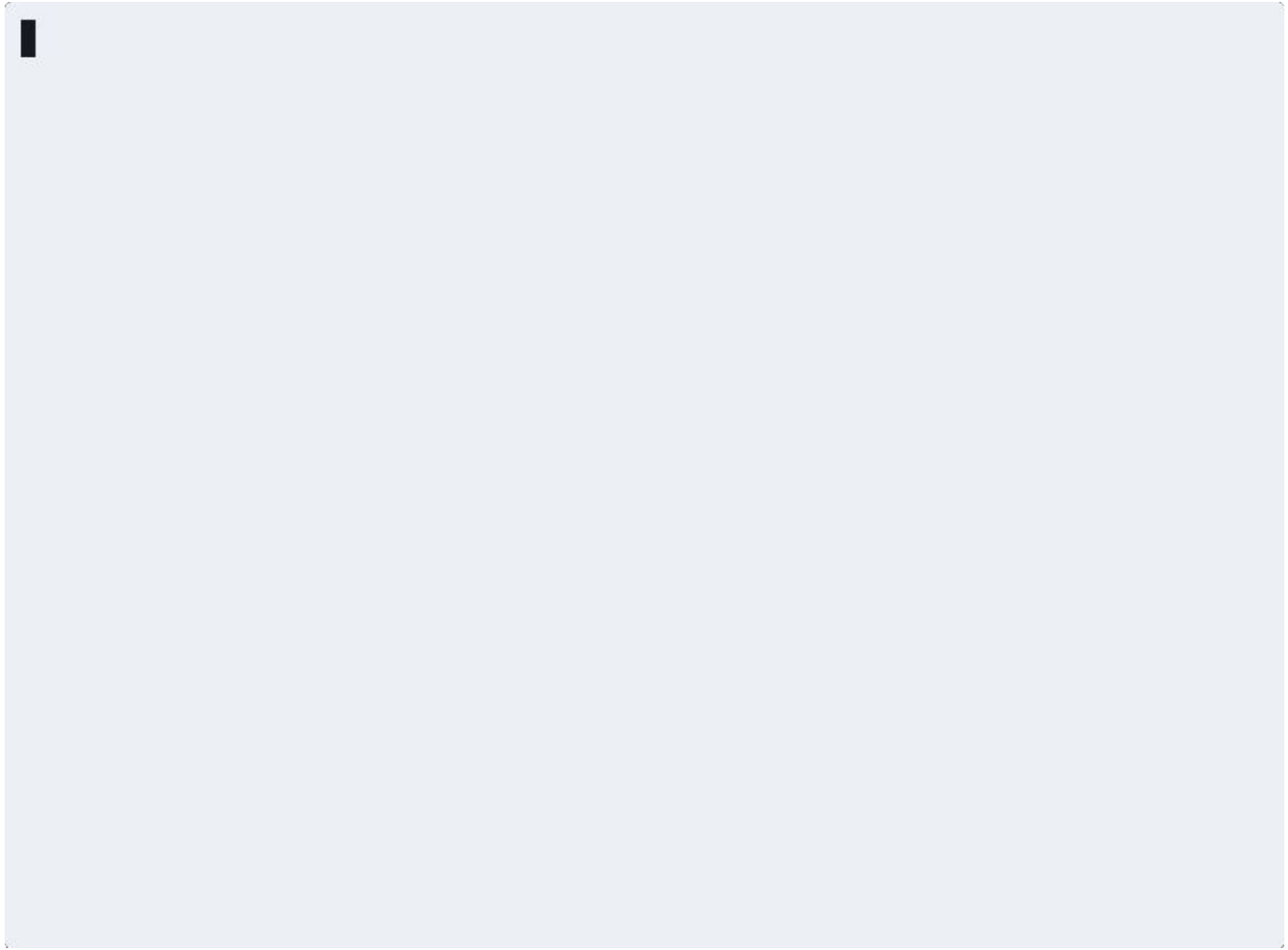
Can we analyze it without enumerating all states?

Epoch #1: Bounded model checking (symbolic)

Explore all paths up to *k* steps, e.g., 20



Kind of symbolic
breadth-first search



Apalache is fast?

Bounded model checking with Apalache

```
$ apalache-mc check
```

3 sec to invariant violation (11 steps)

Explicit-state model checker TLC:

over 40 min on 30 CPU cores

over 3 billion states

400G disk space ⇒ “No space left on device”

(we could do trivial abstraction)

General perception: It is slow

Increasing the number of checkable steps would improve usability #1571

danwt started this conversation in **General**

danwt on Mar 28, 2022

Problem description

In my experience it can be hard
can check can be quite limited.
invariants, and medium/large m
models

Use Z3 in parallel mode #2315

Closed #2990

rnbguy opened on

Is your featu

It would be nice to

For one, I want to

Describe the

Category

General

ITF JSON as init TLA s

Open

rnbguy opened on Mar 3, 2023

Is your feature request related to a problem? Please de

I want to restart model exploration from an ITF trace. I have a ITF JSON state, but
operator in the specification.

Describe the solution you'd like

I would like a checker argument which I can use to pass the ITF JSON string - representing the concrete value assignment of the model state.

spliting trace length to speed up apalache simulate #2419

Open

rnbguy opened on Feb 16, 2023 · edited by rnbguy Edits Contributor

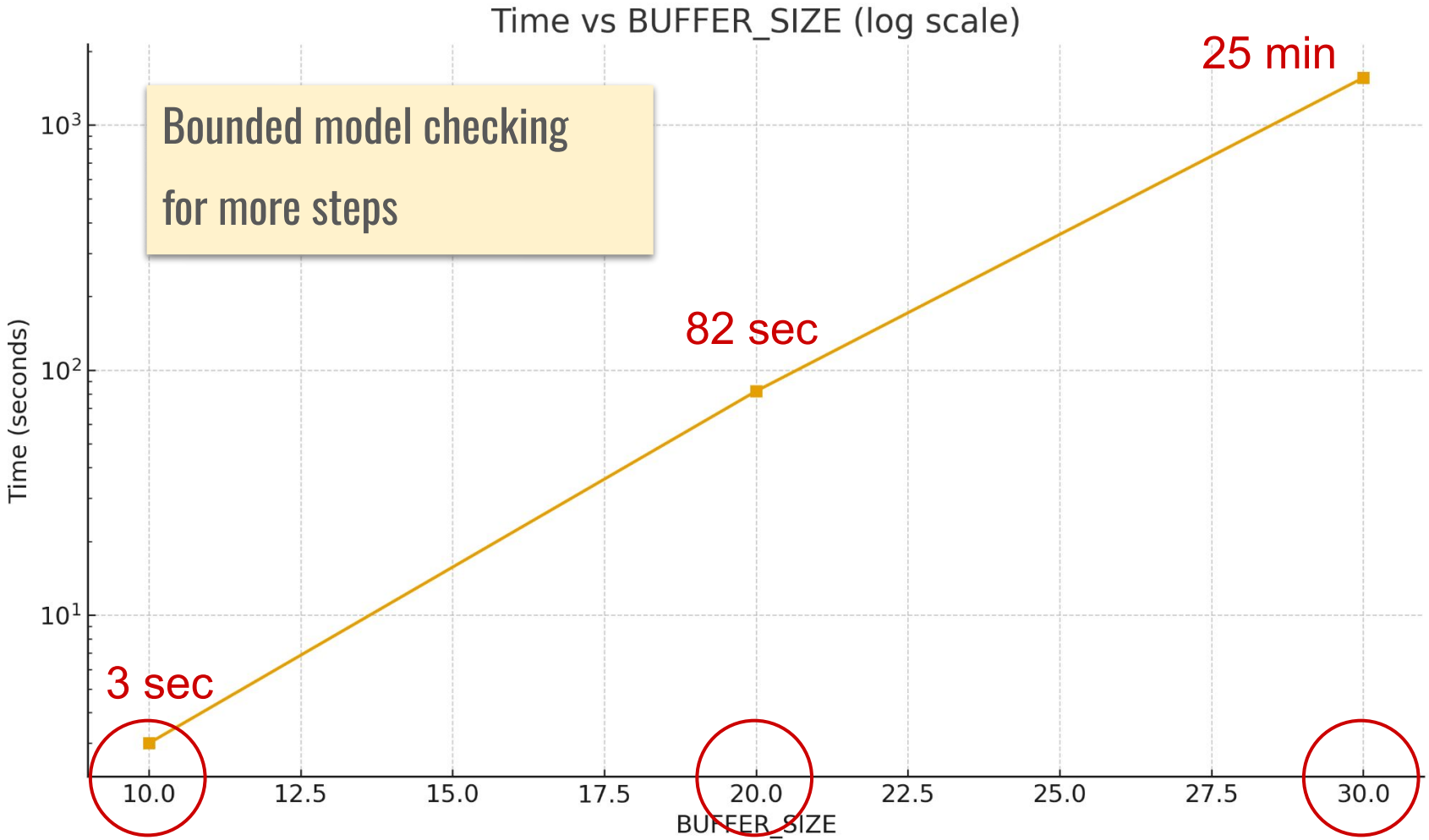
Is your feature request related to a problem? Please describe

Apalache simulate still slow if the model is too big.

Describe the solution you'd like

When I use Apalache to simulate random traces, but I don't care about an invariant being broken, It makes sense to split the traces into multiple short lengths and solve them separately.

We only have to make sure that the `init` operator of each run is the last state operator of the previous run.


Time vs BUFFER_SIZE (log scale)
Bounded model checking
for more steps
25 min
82 sec
3 sec
Time (seconds)
10^3
10^2
10^1
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
BUFFER_SIZE

Weird turn

In 2022-2023, after plenty of discussions with engineers:

I wrote a randomized simulator in TypeScript à la PBT

…and everybody was: “It’s really fast!”

Some of my peers use exclusively this simulator

```
$ quint run --main=mc10u8_buggy_circular_buffer --invariant=safeInv \
    --max-steps=200 buggy_circular_buffer.qnt
…
[violation] Found an issue (16ms at 63 traces/second).
```

Random search?

TLC -simulate -depth 100:

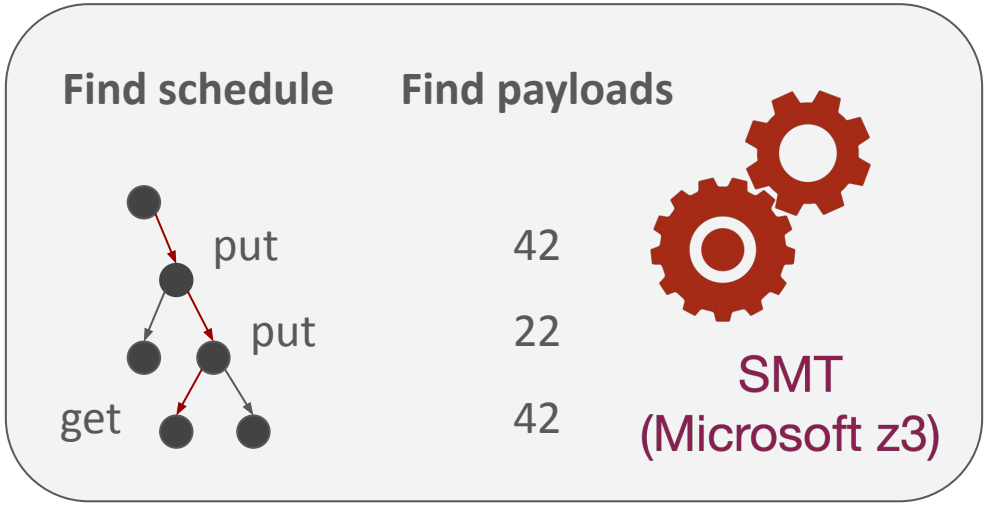
Finds a counterexample in 0.25 sec!

Needs to explore only 174K states

Under 1 sec for BUFFER_SIZE in { 20, 30, 100 }

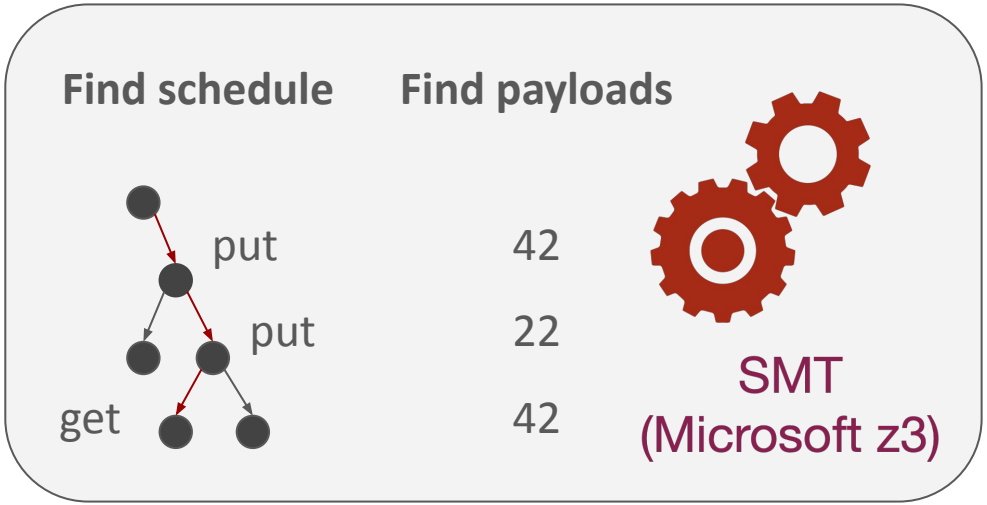
By fixing the search procedure,
we are missing easy bugs!

By fixing the search procedure, we are missing easy bugs!

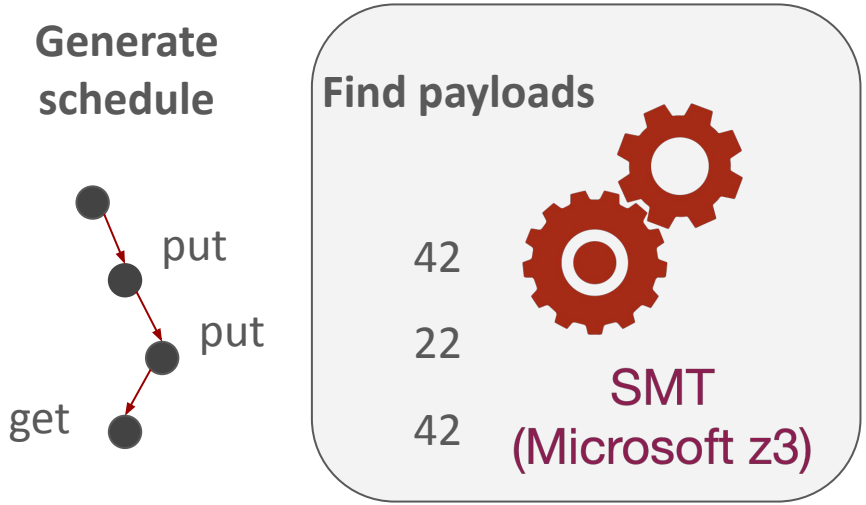


Bounded model checking

Epoch #2: Randomized symbolic execution



Bounded model checking



Randomized symbolic execution

apalache-mc simulate

✅ Finds bugs faster than bounded model checker

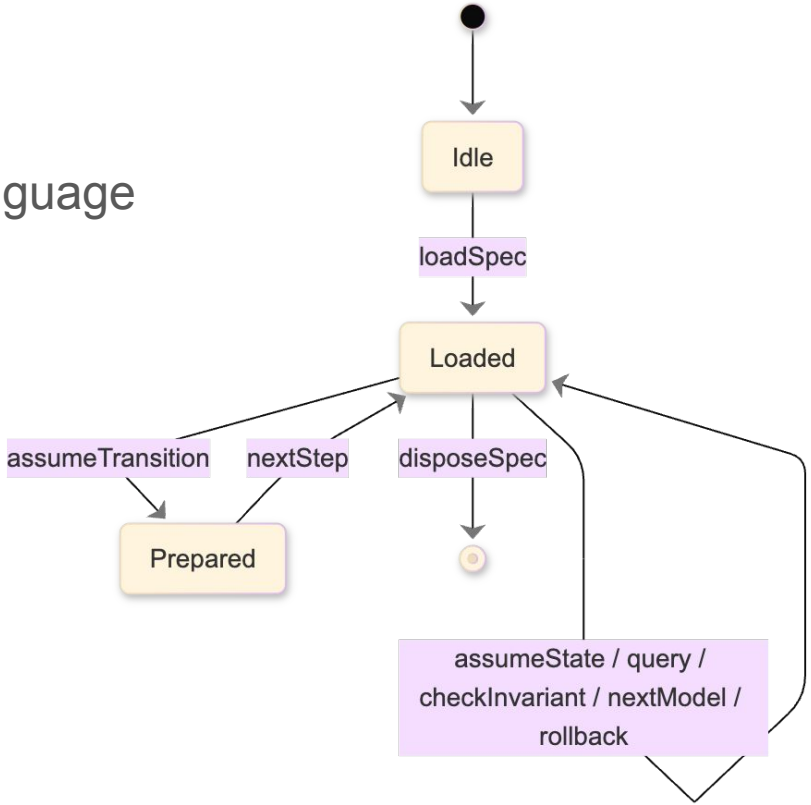
✅ Easier for Z3, as it is mostly doing propagation

✅ Trivially parallelizable

❌ Hard to customize

Epoch #3: DIY search

- ✅ Write your own search scripts in any language
- ✅ Prioritize schedules as you like
- ✅ Evaluate them with Apalache and Z3
- ✅ Query for traces and enumerate models



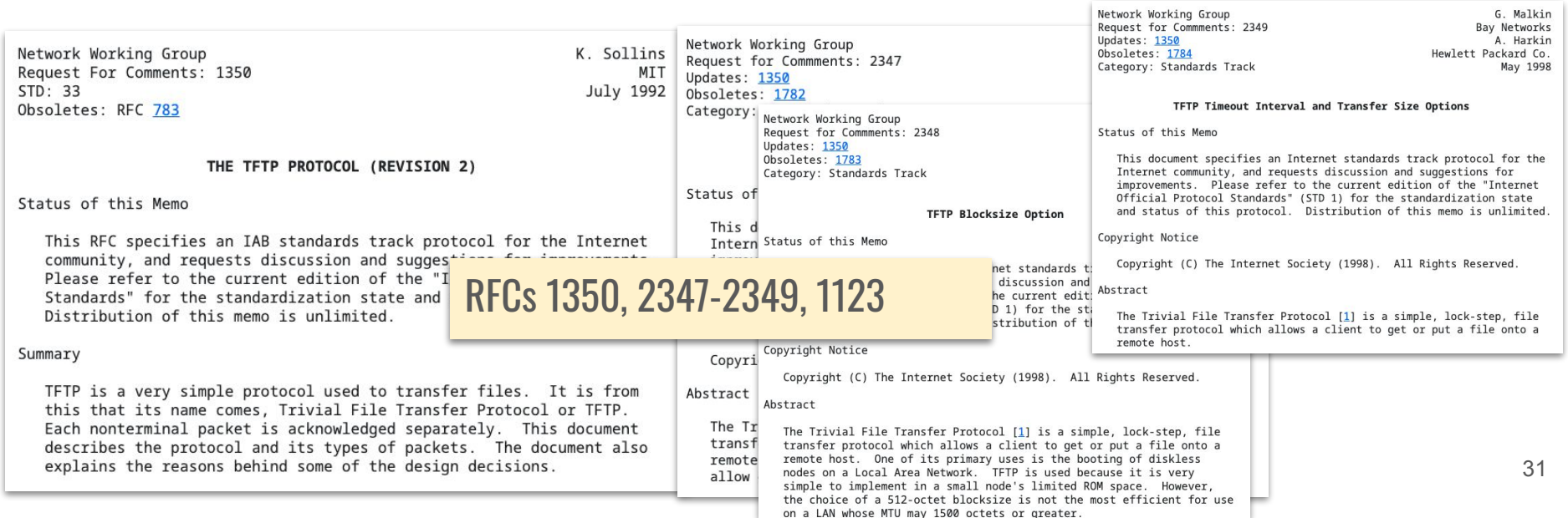
TFTP

Trivial File Transfer Protocol

TFTP

Simple protocol to send and receive files over UDP since 1992

Network booting, transferring firmware to network devices



Network Working Group
Request For Comments: 1350
STD: 33
Obsoletes: RFC 783

K. Sollins
MIT
July 1992

THE TFTP PROTOCOL (REVISION 2)

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

RFCs 1350, 2347-2349, 1123

Network Working Group
Request for Commments: 2347
Updates: 1350
Obsoletes: 1782
Category:

Network Working Group
Request for Commments: 2348
Updates: 1350
Obsoletes: 1783
Category: Standards Track

TFTP Blocksize Option

Status of this Memo

Copyright Notice

Copyright (C) The Internet Society (1998). All Rights Reserved.

Abstract

The Trivial File Transfer Protocol [1] is a simple, lock-step, file transfer protocol which allows a client to get or put a file onto a remote host. One of its primary uses is the booting of diskless nodes on a Local Area Network. TFTP is used because it is very simple to implement in a small node's limited ROM space. However, the choice of a 512-octet blocksize is not the most efficient for use on a LAN whose MTU may 1500 octets or greater.

Network Working Group
Request for Commments: 2349
Updates: 1350
Obsoletes: 1784
Category: Standards Track

G. Malkin
Bay Networks
A. Harkin
Hewlett Packard Co.
May 1998

TFTP Timeout Interval and Transfer Size Options

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

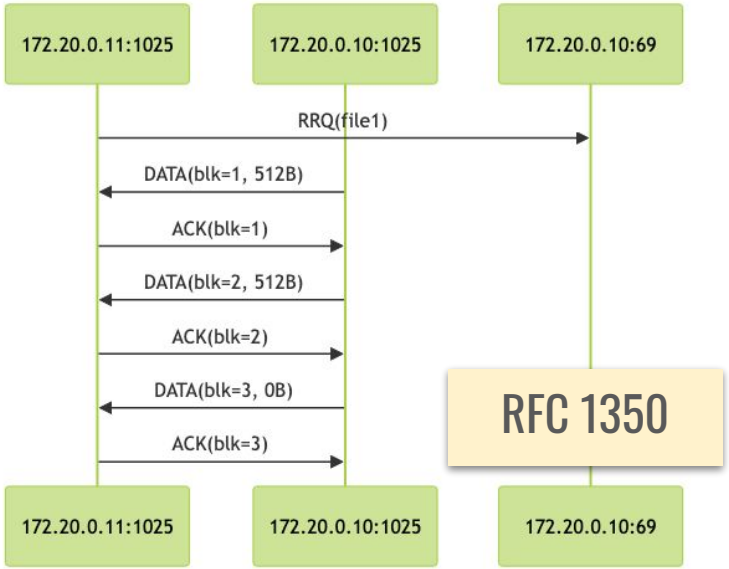
Copyright (C) The Internet Society (1998). All Rights Reserved.

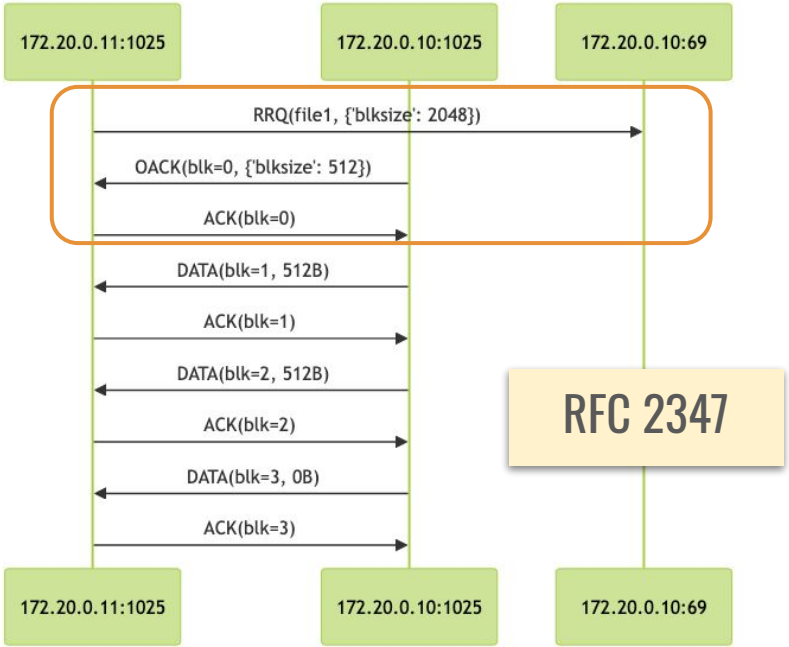
Abstract

The Trivial File Transfer Protocol [1] is a simple, lock-step, file transfer protocol which allows a client to get or put a file onto a remote host.

Happy paths are easy

only read requests in this talk





TFTP challenges for verification tools

1. **UDP**: packet loss, reordering, retransmission
2. **Large parameter space**: 64K of ports, block sizes, file sizes
3. Retransmission **timeouts**
4. Some operations are **input/output**
5. Some are **one-sided** (errors, termination)

TFTP is only called trivial

Specification in TLA$^+$

Initial spec

Client: 5 actions

Server: 5 actions

Environment: 1 action

Size: about 800 LOC

```
Next ==
    \* the actions by the clients
    \/  \E srcIp \in CLIENT_IPS, srcPort \in PORTS:
            \E filename \in DOMAIN FILES, timeout \in 1..255:
                \* "man tftpd": 65464 is the theoretical maximum for block size
                \* https://linux.die.net/man/8/tftpd
                \E tsize \in 0..FILES[filename], blksize \in 0..65464:
                    \* choose a subset of the options to request
                    \E optionKeys \in SUBSET OPTIONS_RFC2349:
                        LET options ==
                            mk_options(optionKeys, blksize, tsize, timeout)
                        IN
                        ClientSendRRQ(srcIp, srcPort, filename, options)
    \/  \E udp \in packets:
            \/ ClientRecvDATA(udp)
            \/ ClientRecvOACK(udp)
            \/ ClientRecvErrorAndCloseConn(udp)
    \/  \E ipPort \in DOMAIN clientTransfers:
            ClientTimeout(ipPort)
    \* the server
    \/  \E udp \in packets:
            \/ ServerRecvRRQ(udp)
            \/ ServerSendDATA(udp)
            \/ ServerRecvAckAndCloseConn(udp)
            \/ ServerRecvErrorAndCloseConn(udp)
    \/  \E ipPort \in DOMAIN serverTransfers:
            ServerTimeout(ipPort)
    \* handle the clock and timeouts
    \/  \E delta \in 1..255:
            AdvanceClock(delta)
```

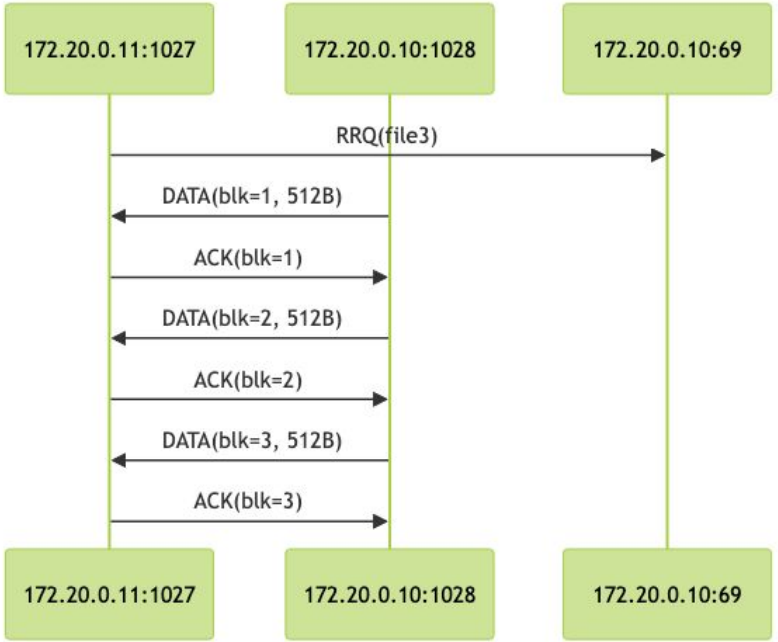
Example: ClientSendRRQ

```
\* A client sends a read request to the server.
\* @type: (Str, Int, Str, Str -> Int) => Bool;
ClientSendRRQ(_srcIp, _srcPort, _filename, _options) ==
    \* We only specify the "octet" mode:
    \* "mail" mode is obsolete as per RFC 1350, what is "net
    ClientSendRRQ::
    LET rrq == RRQ(_filename, "octet", _options)
        udp == [srcIp |-> _srcIp,
                srcPort |-> _srcPort,
                destIp |-> SERVER_IP,
                destPort |-> 69, \* the port is fixed as per
                payload |-> rrq]
    IN
    /\ <<_srcIp, _srcPort>> \notin DOMAIN clientTransfers
    \* RFC-2349: In Read Request packets, a size of "0" is s
    /\ "tsize" \in DOMAIN _options => _options["tsize"] = 0
```

```
/\ clientTransfers' = [
        p \in DOMAIN clientTransfers \union {<<_srcIp, _srcPort>>} |->
        IF p = <<_srcIp, _srcPort>>
        THEN [ port |-> 69, \* initial port before negotiation
               tsize |-> 0,
               blksize |-> get_or_else(_options, "blksize", 512),
               timeout |-> get_or_else(_options, "timeout", 1),
               blocks |-> <<>>,
               blockNum |-> 0,
               timestamp |-> clock,
               transferred |-> 0,
               proto |->
                IF DOMAIN _options = {}
                THEN PROTO_OPTIONS_NO ELSE PROTO_OPTIONS_YES
             ]
        ELSE clientTransfers[p]
    ]
/\ packets' = packets \union {udp}
/\ lastAction' = ActionClientSendRRQ(udp)
/\ UNCHANGED <<serverTransfers, clock>>
```

Make sure the spec does something

```
$ apalache-mc check \
  --inv=RecvThreeDataBlocksEx MC2_tftp.tla
...
State 7: state invariant 0 violated.
Total time: 24.813 sec
```




```
\* Check this falsy invariant to see an example
RecvThreeDataBlocksEx ==
    ~(\E p \in DOMAIN clientTransfers:
        Len(clientTransfers[p].blocks) >= 3)
```

Initial specification effort

1. Relatively straightforward with my experience
2. Most of my time went into switching between RFCs
3. Generating repetitive type definitions with LLMs
4. Playing with falsy invariants to produce examples

[github.com/konnov/tftp-symbolic-testing]

13 hours ≈ 2 days

Question #1 from every engineer:

How does it connect to the implementation?

Tester-implementation game

Player 1 (tester)

1. Guess next transition with inputs

3. Evaluate the response. Repeat 1.

Player 2 (SUT)

2. Execute the transition. Respond.

Multishot

input-output conformance testing

Challenges

1. Completely random generation is really low quality
2. Produce high-quality inputs
3. Develop the harness and testing infrastructure

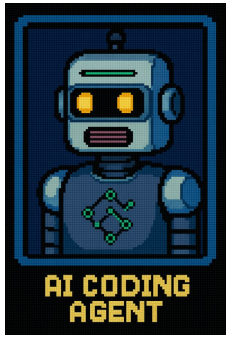
1. Completely random generation is really low quality
2. Produce high-quality inputs

3. Develop the harness and testing infrastructure

Give me a framework in Golang, Rust, etc.

I don't want to learn TLA$^+$ and Python

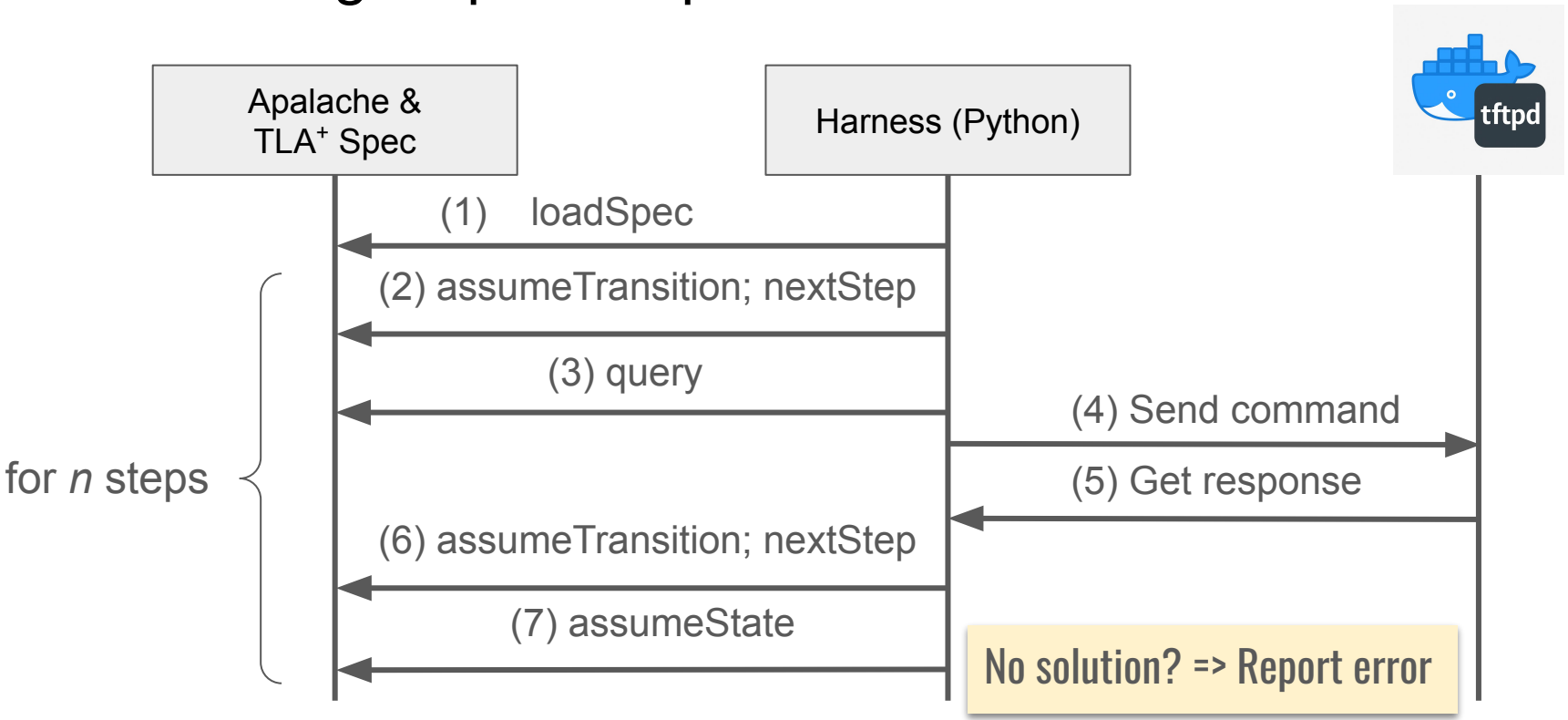
I'm coding



Sure, I will help you to write the test harness!

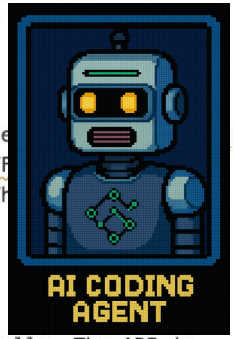
(You have to fix my bugs later. He-he)

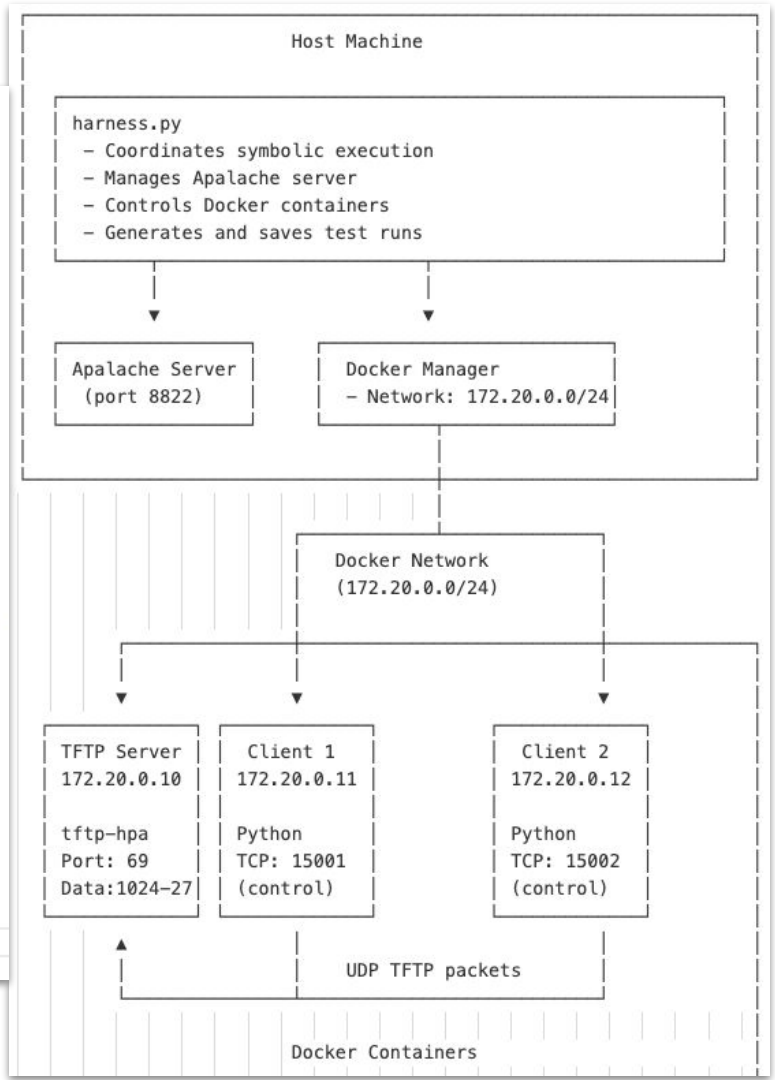
New testing loop with Apalache



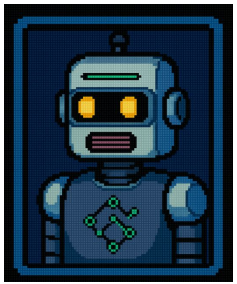
prompt-test-harness.md > ## Requirement 2

```
1  You are a protocol testing engineer. Your goal is to use the TLA+ specification
2  of TFTP to create a test harness that produces new tests by analyzing symbolic
3  executions of the protocol via JSON RPC of Apalache (see [client.py][]).
4
5  Below are the requirements of this project. Do not change this text unless
6  explicitly instructed to do so.
7
8  ## Requirement 0
9
10 The TLA+ specification of the TFTP protocol is provided in the [spec][] file.
11 The test harness should utilize this specification to generate symbolic
12 execution traces using Apalache. You are not allowed to modify the TLA+
13 specification, unless you are explicitly instructed to do so.
14
15 ## Requirement 1
16
17 The test harness must be implemented in Python and should inte
18 using JSON RPC to retrieve symbolic execution traces of the TF
19 Python script is located at [test-harness](./test-harness). Th
20 should use Poetry to pull in dependencies.
21
22 ## Requirement 2
23
24 The communication with Apalache must be done using JSON RPC calls. The API is
25 implemented in the [client.py][] file, which provides functions to send requests
26 to Apalache and receive responses. The harness should utilize these functions to
27 obtain symbolic execution traces for the TFTP protocol.
28
29 ## Requirement 3
```





Debugging the harness



≈ 1 week
≈ 200 premium requests (Copilot)

1. Lots of debugging
2. Explaining Claude how to handle packets
3. Analyzing logs
4. Fixing bugs in the generated harness code
5. Claude lost in abstractions (e.g., clients in docker vs. clients in spec)

How it looks like

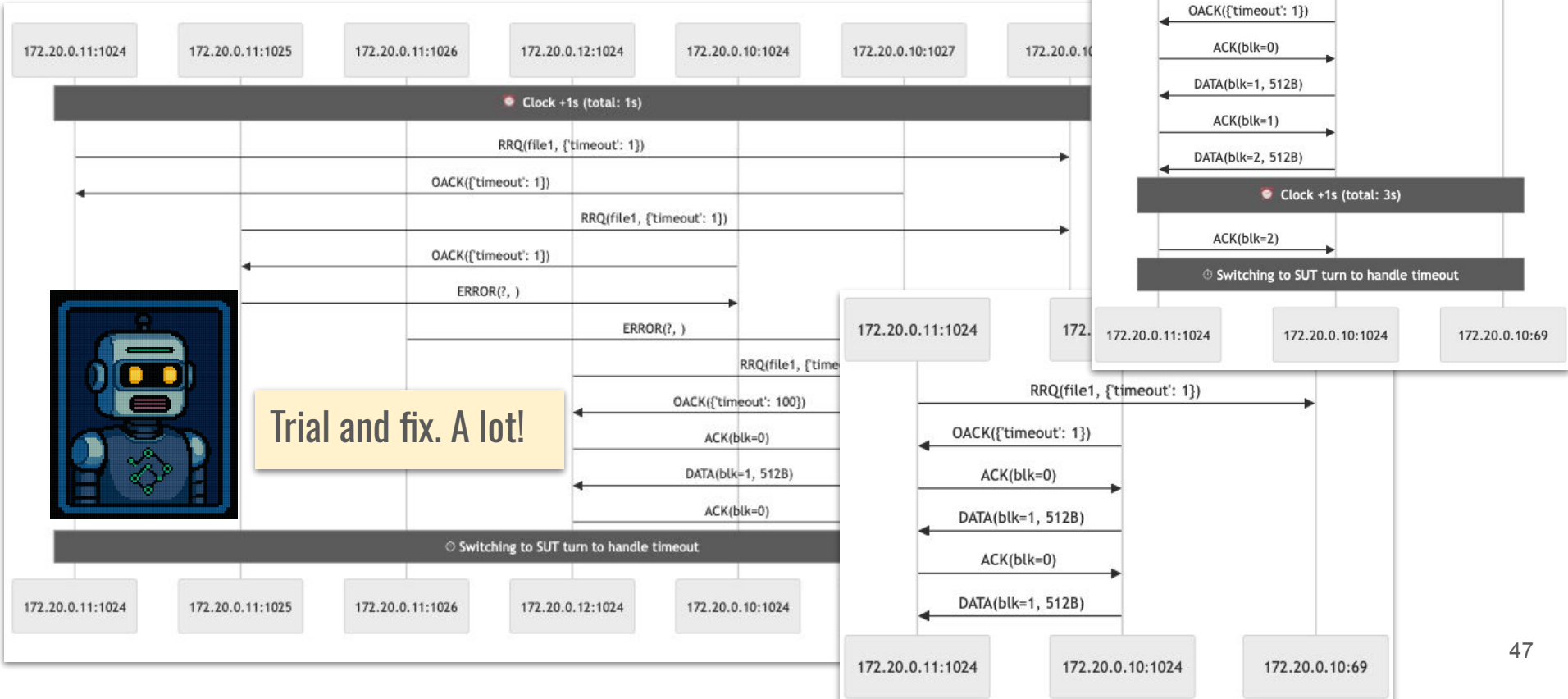
python_harness.log

```
2025-11-12 22:21:03,033 - __main__ - INFO - === Starting test run generation 2 (max 100 steps) ===
2025-11-12 22:21:03,033 - __main__ - INFO - Trying transition 0 [Init]...
2025-11-12 22:21:03,035 - client - INFO - Transition 0: ENABLED
2025-11-12 22:21:03,035 - __main__ - INFO - Transition 0 is ENABLED
2025-11-12 22:21:03,036 - client - INFO - Moved to step 1
2025-11-12 22:21:03,036 - __main__ - INFO -
--- Step 1/100 ---
2025-11-12 22:21:03,036 - __main__ - INFO - Trying transition 4 [ClientRecvDATA]...
2025-11-12 22:21:03,038 - client - INFO - Transition 4: DISABLED
2025-11-12 22:21:03,038 - __main__ - INFO - Transition 4 is DISABLED
2025-11-12 22:21:03,038 - __main__ - INFO - Rollback to snapshot 3
2025-11-12 22:21:03,039 - __main__ - INFO - Trying transition 5 [ClientRecvOACKthenSendAck]...
2025-11-12 22:21:03,040 - client - INFO - Transition 5: DISABLED
2025-11-12 22:21:03,040 - __main__ - INFO - Transition 5 is DISABLED
2025-11-12 22:21:03,040 - __main__ - INFO - Rollback to snapshot 3
2025-11-12 22:21:03,041 - __main__ - INFO - Trying transition 3 [AdvanceClock]...
2025-11-12 22:21:03,042 - client - INFO - Transition 3: ENABLED
2025-11-12 22:21:03,042 - __main__ - INFO - Transition 3 is ENABLED
2025-11-12 22:21:03,042 - client - INFO - Moved to step 2
2025-11-12 22:21:03,042 - __main__ - INFO - Executing TFTP operation for transition 3
2025-11-12 22:21:03,044 - __main__ - INFO - Retrieved trace with 2 states
2025-11-12 22:21:03,044 - __main__ - INFO - Action: Advance Clock by 1
2025-11-12 22:21:03,044 - __main__ - INFO -   Sleeping for 1 seconds to advance clock...
2025-11-12 22:21:04,049 - __main__ - INFO -   ✓ Clock advanced by 1 seconds
2025-11-12 22:21:04,049 - __main__ - INFO -
--- Step 2/100 ---
2025-11-12 22:21:04,050 - __main__ - INFO - Trying transition 4 [ClientRecvDATA]...
2025-11-12 22:21:04,053 - client - INFO - Transition 4: DISABLED
2025-11-12 22:21:04,053 - __main__ - INFO - Transition 4 is DISABLED
2025-11-12 22:21:04,053 - __main__ - INFO - Rollback to snapshot 6
2025-11-12 22:21:04,055 - __main__ - INFO - Trying transition 6 [ClientRecvOACKthenSendError]...
2025-11-12 22:21:04,057 - client - INFO - Transition 6: DISABLED
2025-11-12 22:21:04,057 - __main__ - INFO - Transition 6 is DISABLED
2025-11-12 22:21:04,057 - __main__ - INFO - Rollback to snapshot 6
2025-11-12 22:21:04,059 - __main__ - INFO - Trying transition 0 [ClientSendRRQ]...
2025-11-12 22:21:04,066 - client - INFO - Transition 0: ENABLED
2025-11-12 22:21:04,066 - __main__ - INFO - Transition 0 is ENABLED
2025-11-12 22:21:04,067 - client - INFO - Moved to step 3
```

tftpd_server.log

```
TFTP server watchdog started with PID 7
Started rsyslog daemon
Starting TFTP server (attempt 1/1)
Command: in.tftpd -vvv --foreground --address 172.20.0.10:69 --user nobody -
4:1027 --secure /var/tftp
Started tcpdump with PID 11 (unbuffered output)
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot
ytes
21:21:00.532140 eth0  In  IP (tos 0x0, ttl 64, id 11761, offset 0, flags [DF
7), length 52)
    172.20.0.11.1024 > 172.20.0.10.69: [bad udp cksum 0x586f -> 0x514d!] TFT
RQ "file1" octet timeout 1
21:21:00.533368 eth0  Out IP (tos 0x0, ttl 64, id 23847, offset 0, flags [no
(17), length 40)
    172.20.0.10.1027 > 172.20.0.11.1024: [bad udp cksum 0x5863 -> 0xa93a!] U
21:21:00.582224 eth0  In  IP (tos 0x0, ttl 64, id 52516, offset 0, flags [DF
7), length 64)
    172.20.0.12.1025 > 172.20.0.10.69: [bad udp cksum 0x587c -> 0x4ba7!] TFT
RQ "file1" octet blksize 513 timeout 2
21:21:00.583130 eth0  Out IP (tos 0x0, ttl 64, id 33392, offset 0, flags [no
(17), length 52)
    172.20.0.10.1024 > 172.20.0.12.1025: [bad udp cksum 0x5870 -> 0xa397!] U
21:21:00.628954 eth0  In  IP (tos 0x0, ttl 64, id 11822, offset 0, flags [DF
7), length 32)
    172.20.0.11.1024 > 172.20.0.10.1027: [bad udp cksum 0x585b -> 0x9f91!] U
21:21:00.629025 eth0  Out IP (tos 0x0, ttl 64, id 23848, offset 0, flags [no
(17), length 544)
    172.20.0.10.1027 > 172.20.0.11.1024: [bad udp cksum 0x5a5b -> 0xa8be!] U
21:21:00.686941 eth0  In  IP (tos 0x0, ttl 64, id 52596, offset 0, flags [DF
7), length 58)
```

Generating sequence diagrams



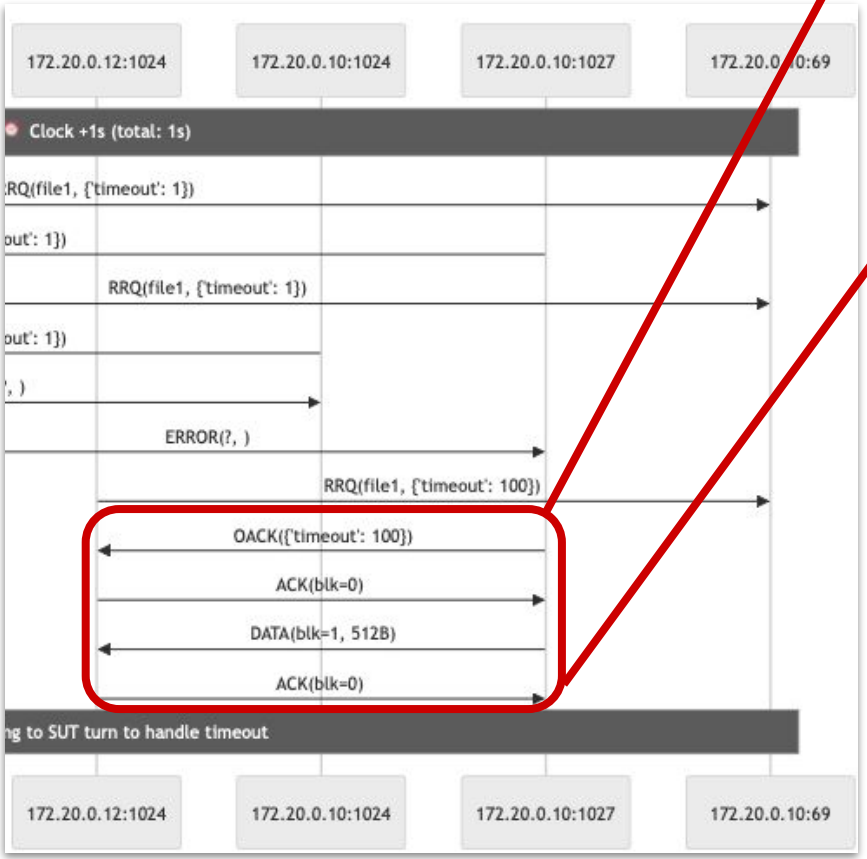
Spec vs. Code (1)

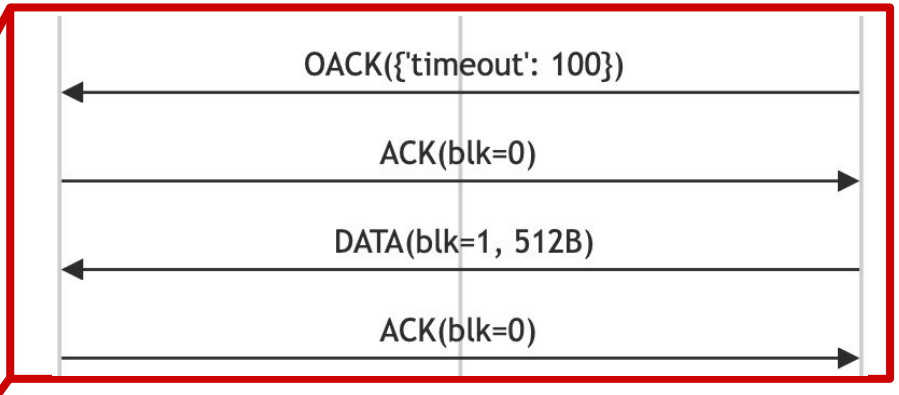
RFC 2347: "*...the server should simply omit the option from the OACK, respond with an alternate value, or send an ERROR packet, with error code 8, to terminate the transfer.*"

The server sends ERROR from the control port (69). Nope.

```
...↑...   @@ -207,7 +207,7 @@ _ServerSendErrorOnRrq(_rrq, _clientIpAndPort, _newServerPort, _rcvdPacket) ==
207  207        ServerRecvRRQthenSendError::
208  208        \E errorCode \in DOMAIN ALL_ERRORS:
209  209            LET errorPacket == [
210       -                srcIp |-> SERVER_IP, srcPort |-> 69,
     210 +                srcIp |-> SERVER_IP, srcPort |-> _newServerPort,
211  211                  destIp |-> _clientIpAndPort[1], destPort |-> _clientIpAndPort[2],
212  212                  payload |-> ERROR(errorCode)
213  213              ]
```

Harness vs. Code (2)





The client has not received DATA(1) yet and sends ACK(0) again

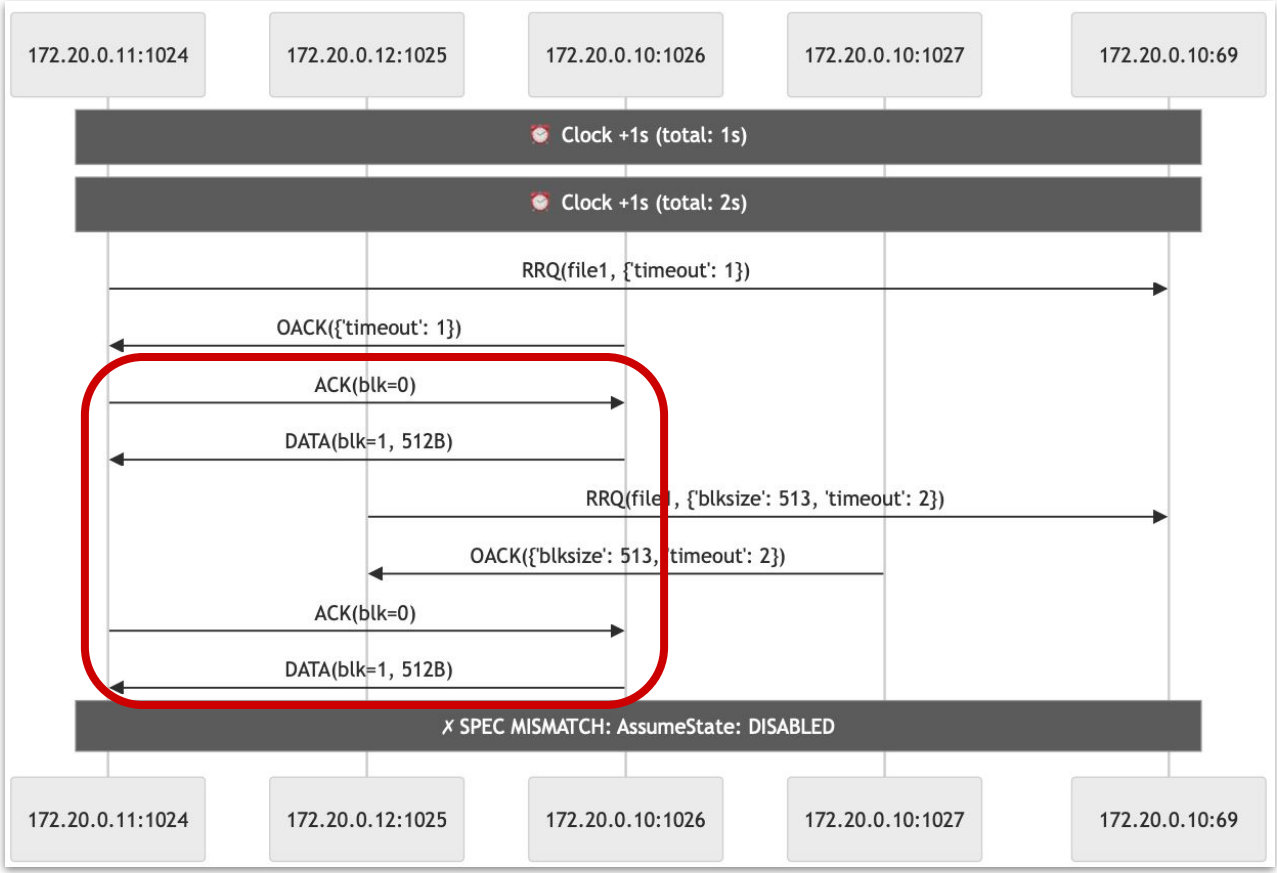
The harness expects a timeout

Fix #2: Give the client a chance to receive DATA after timeout (UDP is hard!)

Spec vs. Code (3)

Our server spec did not send duplicate packets!

UDP is hard 🥺



```
\* The server receives an ACK packet and resends DATA that it sent in the past.
\* This is to fix the mismatch found by the test harness.
\* @type: $udpPacket => Bool;
ServerResendDATA(_udp) ==
    ServerResendDATA::
    LET ipPort == <<_udp.srcIp, _udp.srcPort>> IN
    /\ IsACK(_udp.payload)
    /\ _udp.destIp = SERVER_IP
    /\ ipPort \in DOMAIN serverTransfers
    /\  \E dataPacket \in packets:
        LET ack == AsACK(_udp.payload)
            data == AsDATA(dataPacket.payload)
            transfer == serverTransfers[ipPort]
            \* update the timestamp of the last transfer
            newTransfer == [ transfer EXCEPT !.timestamp = clock ]
        IN
        \* make sure that we receive from the correct port
        /\ _udp.destPort = transfer.port
        \* The DATA packet is sent in response to the ACK.
        /\ ack.blockNum + 1 = data.blockNum
        /\ dataPacket.srcIp = SERVER_IP
        /\ dataPacket.srcPort = _udp.destPort
        /\ dataPacket.destIp = _udp.srcIp
        /\ dataPacket.destPort = _udp.srcPort
        \* do not receive packets if the connection must timeout
        /\ clock <= transfer.timestamp + transfer.timeout
        \* either we have more data to send, or we send exactly 0 bytes in the last block
        /\ serverTransfers' = [ serverTransfers EXCEPT ![ipPort] = newTransfer ]
        /\ lastAction' = ActionRecvSend(_udp, dataPacket)
    /\ UNCHANGED <<packets, clientTransfers, clock>>
```

FIX #3:

Let the server re-send DATA

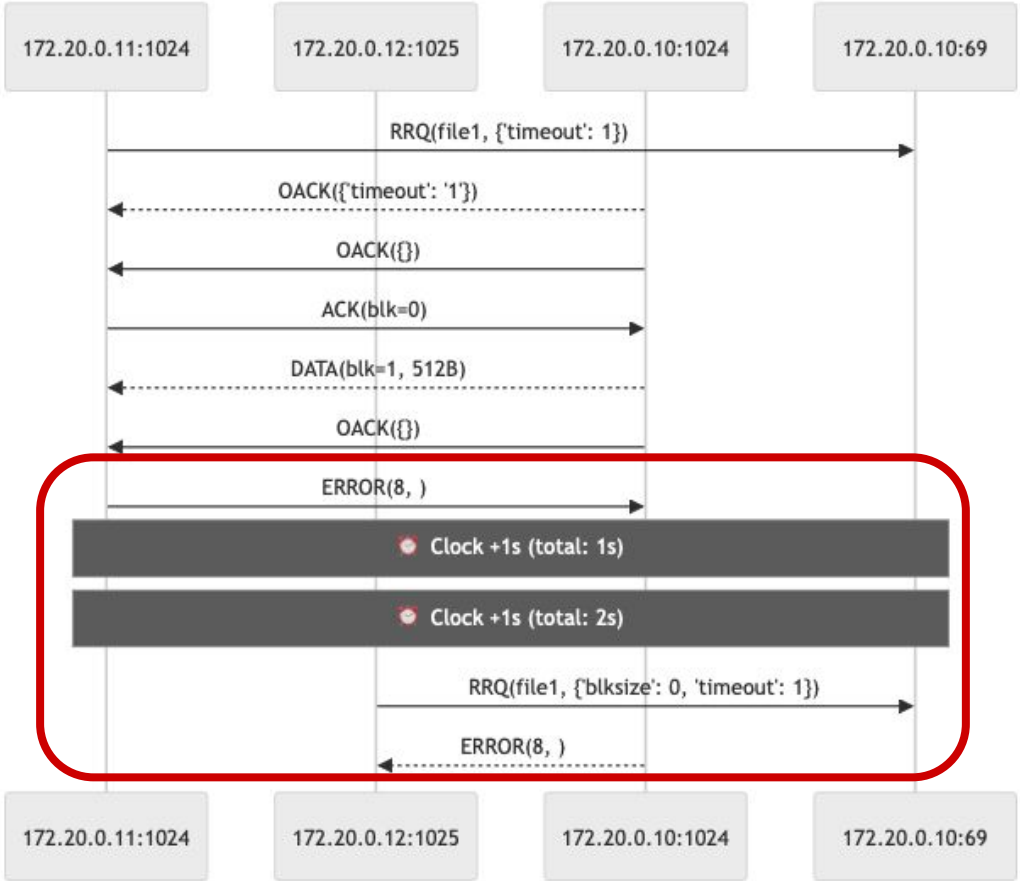
Spec vs. Code (4)

Per RFC 1350, the server does not have to reply on ERROR!

There is input, but no output!

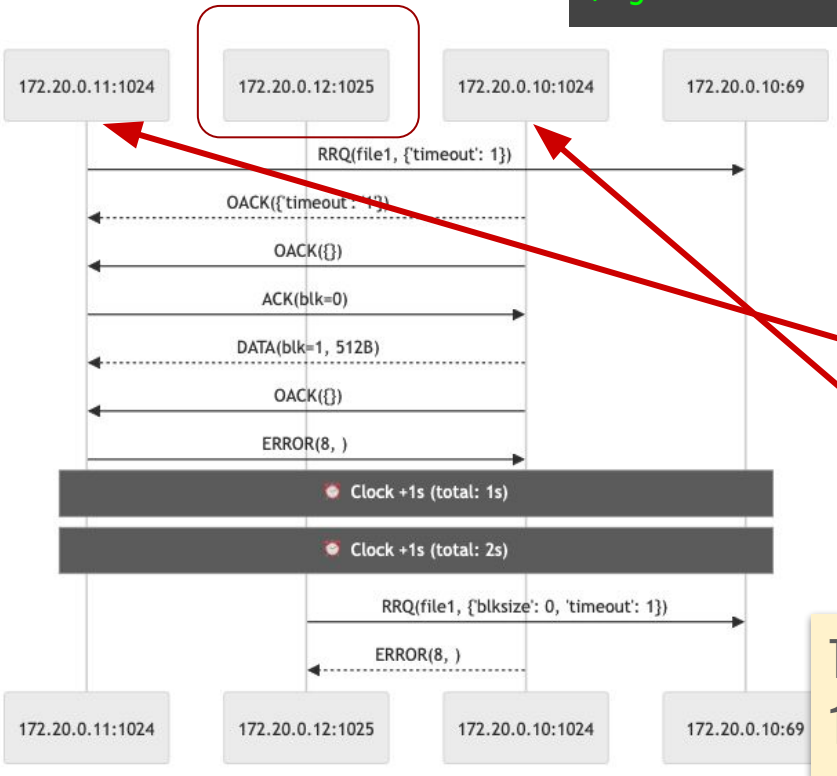
The server in the spec had not received ERROR so far

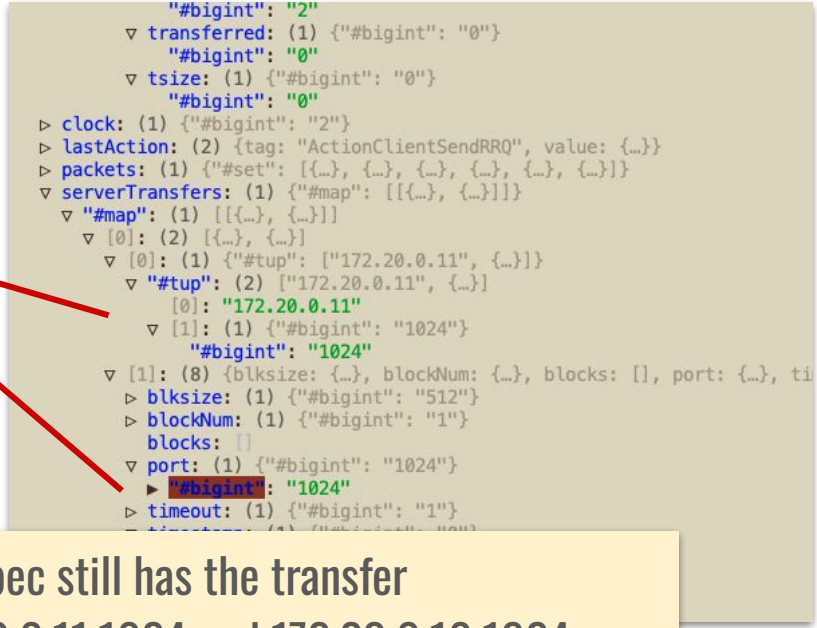
Oops. Our generated harness was supposed to work this way



How do we debug this divergence?

```
$ jless test-results/run_0002/divergence_trace.itf.json
```



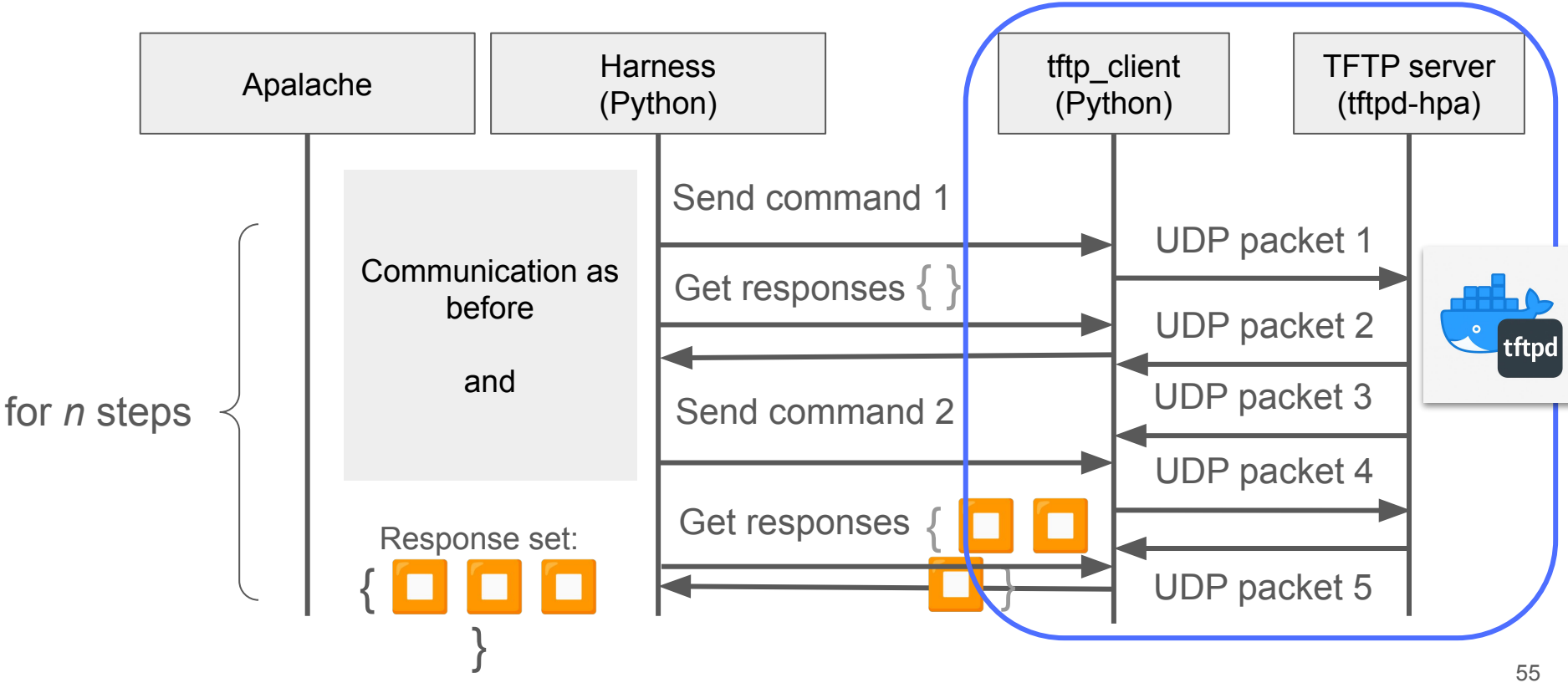


The spec still has the transfer 172.20.0.11:1024 and 172.20.0.10:1024

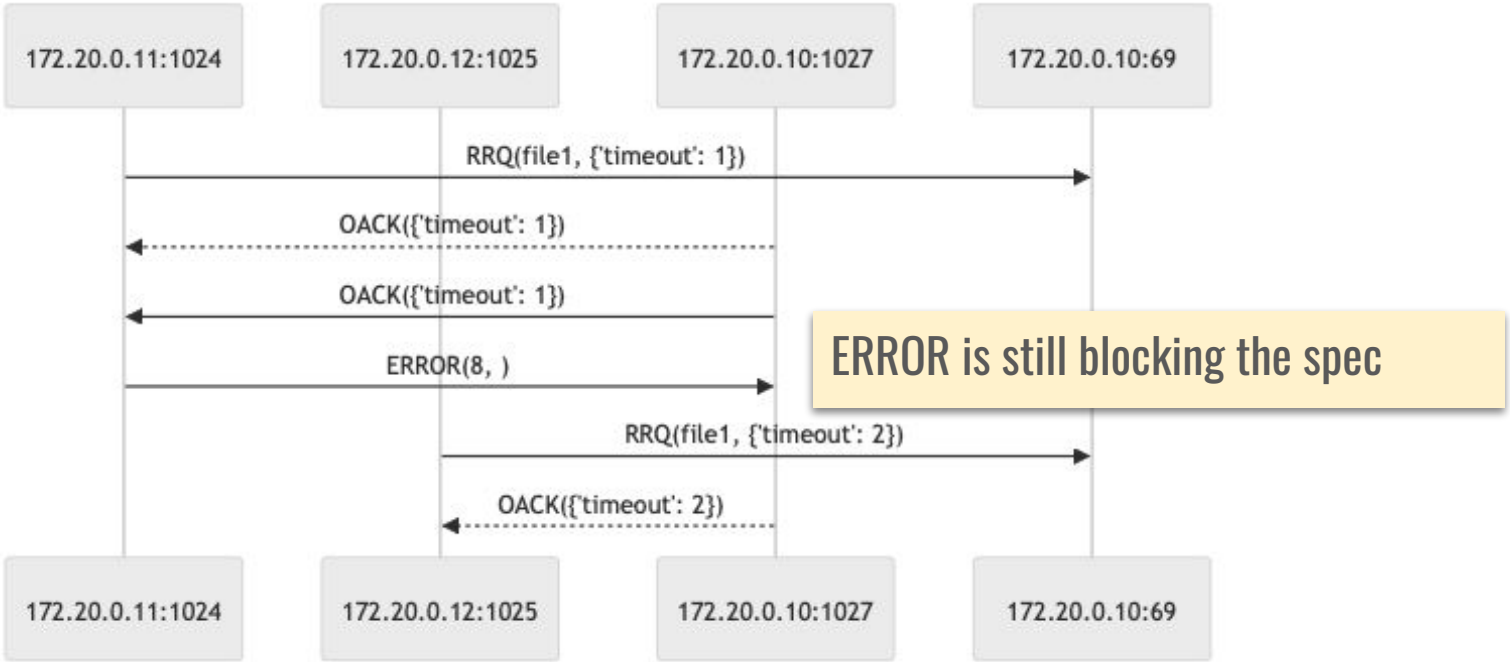
RFC 2347: "*In order to create a connection, each end of the connection chooses a TID for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low.*"

We have hit the case of low probability?

Revised testing loop with Apalache



Does this work?



ERROR is still blocking the spec

FIX #5: the server may reuse the port if there was an ERROR

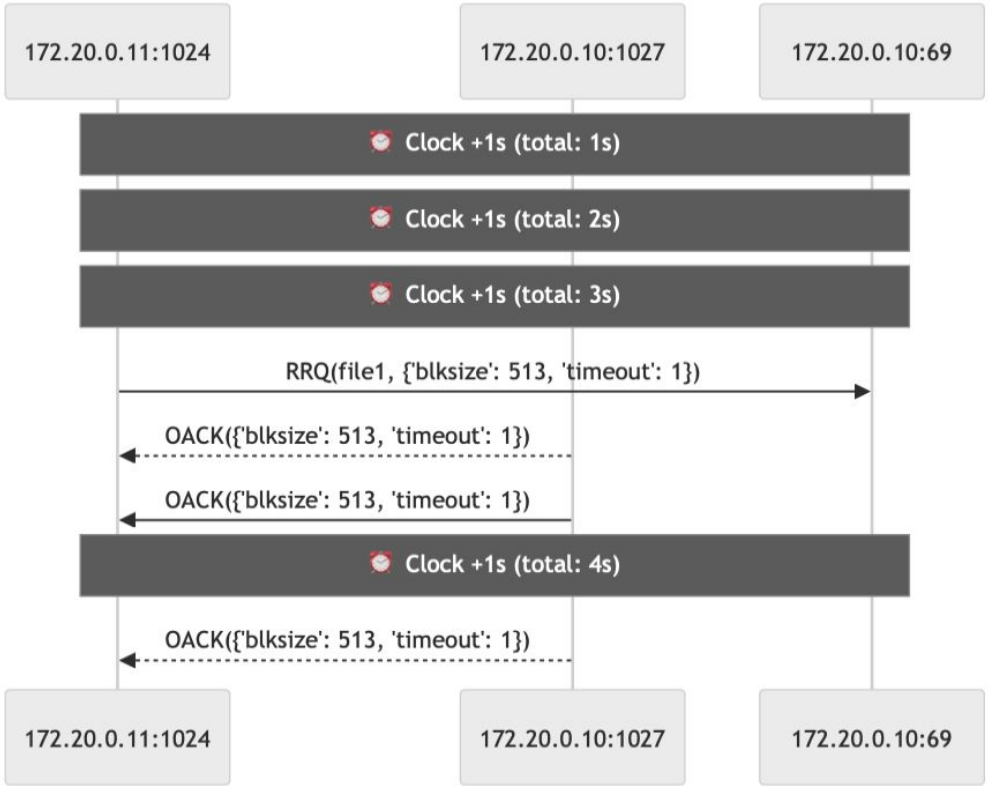
```
\* the server allocates a new port for the connection, if it can find one
/\ \E newServerPort \in PORTS:
    /\  \/  \A p \in DOMAIN serverTransfers:
                serverTransfers[p].port /= newServerPort
        \* Or, there was an ERROR packet that cancelled the active transfer.
        \* This has a bad smell, but is needed to conform tftpd-hpa.
        \/ \E packet \in packets:
            /\ IsERROR(packet.payload)
            /\ packet.destIp = SERVER_IP
            /\ packet.destPort = newServerPort
```

```
\* The server receives RRQ and sends one of: DATA, OACK, or ERROR.
\* @type: $udpPacket => Bool;
ServerRecvRRQ(_udp) ==
    /\ IsRRQ(_udp.payload)
    /\ _udp.destIp = SERVER_IP
    /\ _udp.destPort = 69
    /\  LET rrq == AsRRQ(_udp.payload)
            clientIpAndPort == <<_udp.srcIp, _udp.srcPort>> IN
        \* A client can open multiple connections from different ports.
                                                                ers
                                                                nnection, if it can find one
                                                                wServerPort
                                                                 cancelled the active transfer.
                \* According to RFC 2347, the server may respond with DATA or OACK
                /\  \/ _ServerSendDataOnRrq(rrq, clientIpAndPort, newServerPort, _udp)
                    \/ _ServerSendOackOnRrq(rrq, clientIpAndPort, newServerPort, _udp)
                    \/ _ServerSendErrorOnRrq(rrq, clientIpAndPort, newServerPort, _udp)
```

Easy in TLA+, hard in stricter specification languages!

Spec vs. Code (5)

Duplicate and outdated packets!

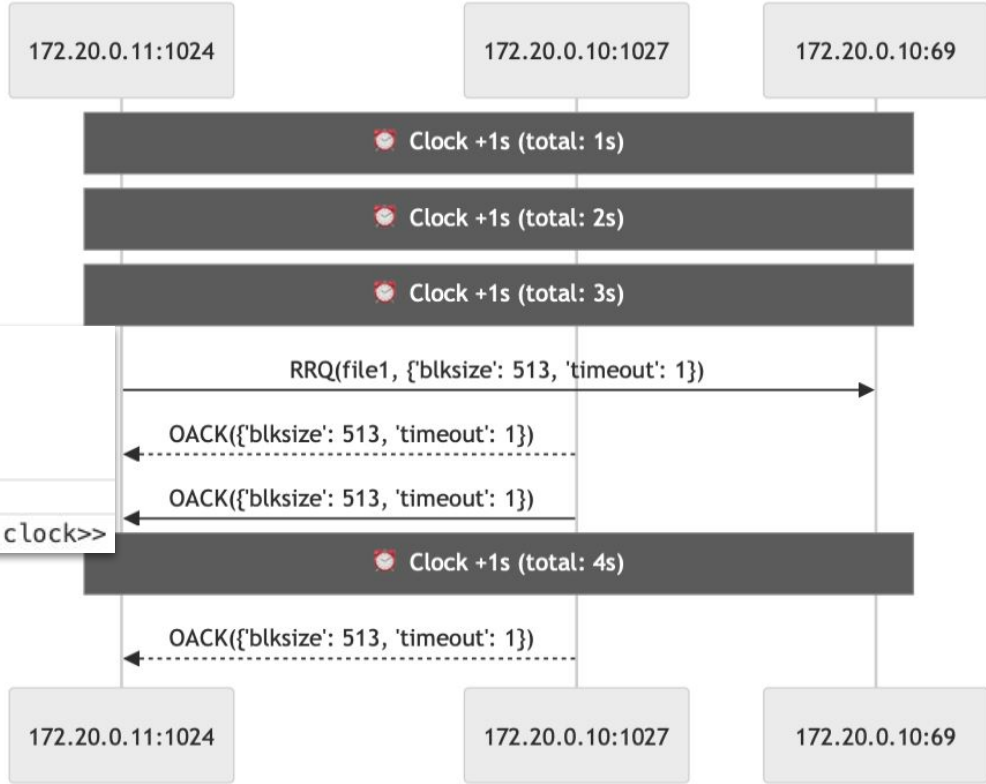


Spec vs. Code (5)

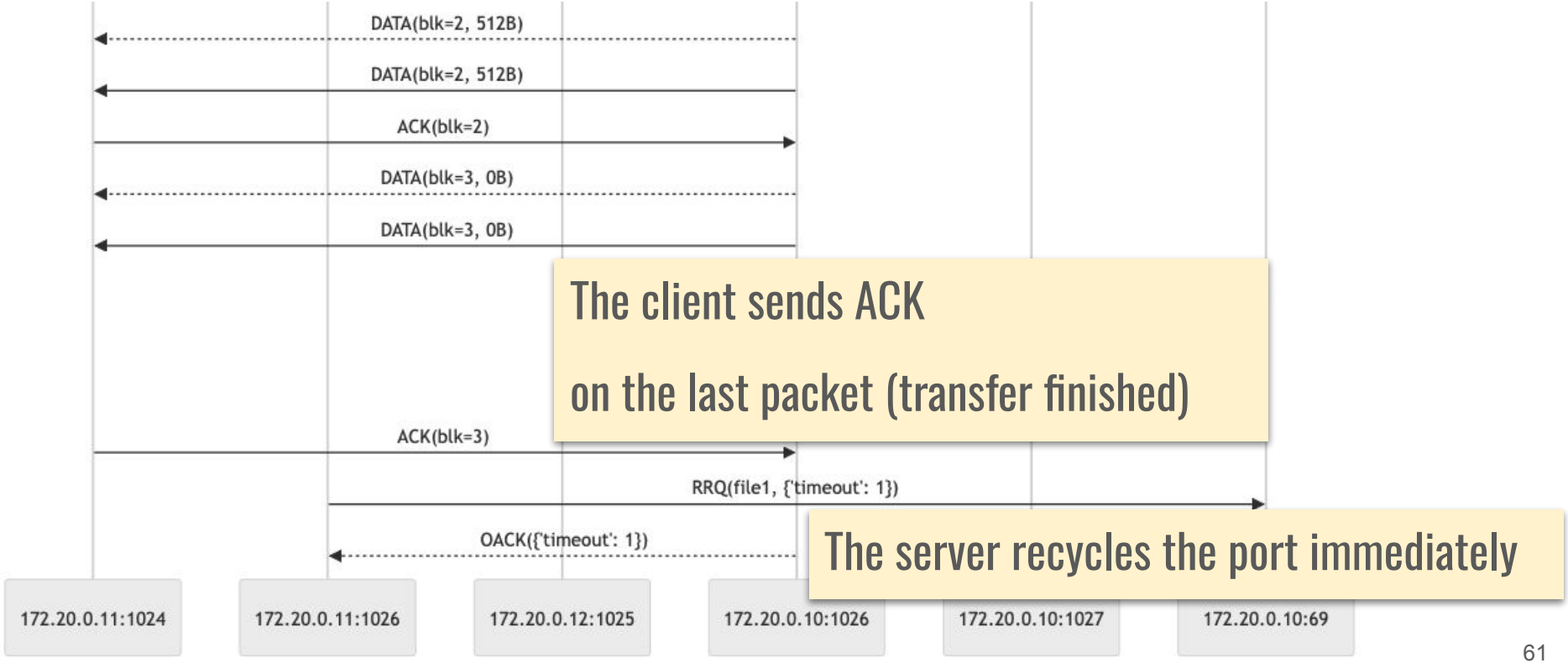
Duplicate and outdated packets!

```
\* @type: $udpPacket => Bool;
ServerSendDup(_udp) ==
    ServerSendDup::
    /\ _udp.srcIp = SERVER_IP
    /\ lastAction' = ActionRecvSend(_udp)
    /\ UNCHANGED <<packets, serverTransfers, clientTransfers, clock>>
```

In TLA$^+$ theory, this is just a stuttering step

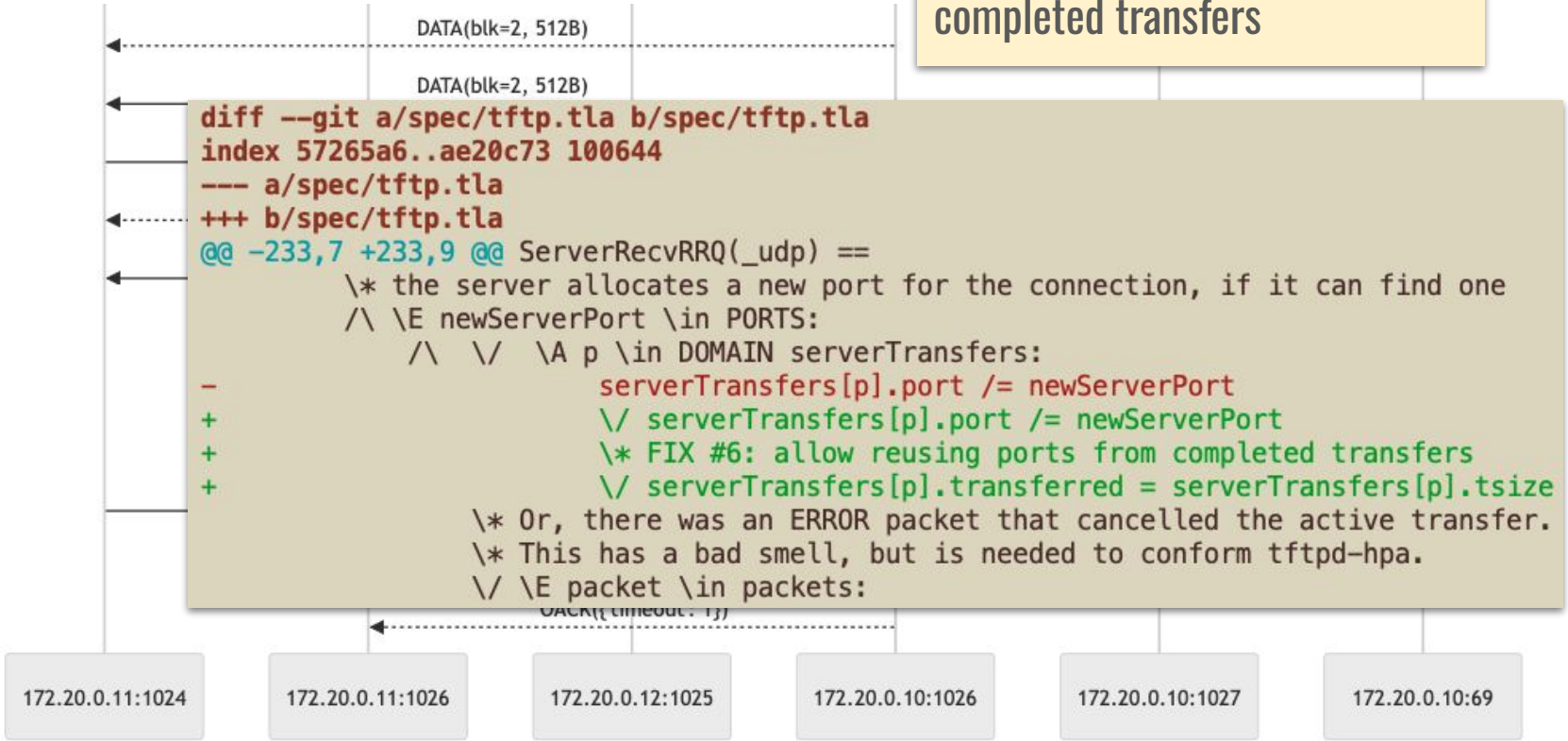


Spec vs. Code (5)



RFC 2347: "*The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e., Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.*"

FIX #6: reuse the ports from completed transfers



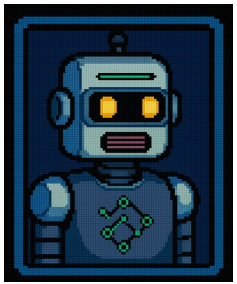

```
diff --git a/spec/tftp.tla b/spec/tftp.tla
index 57265a6..ae20c73 100644
--- a/spec/tftp.tla
+++ b/spec/tftp.tla
@@ -233,7 +233,9 @@ ServerRecvRRQ(_udp) ==
          \* the server allocates a new port for the connection, if it can find one
          /\ \E newServerPort \in PORTS:
              /\  \/  \A p \in DOMAIN serverTransfers:
-                         serverTransfers[p].port /= newServerPort
+                         \/ serverTransfers[p].port /= newServerPort
+                         \* FIX #6: allow reusing ports from completed transfers
+                         \/ serverTransfers[p].transferred = serverTransfers[p].tsize
                 \* Or, there was an ERROR packet that cancelled the active transfer.
                 \* This has a bad smell, but is needed to conform tftpd-hpa.
                 \/ \E packet \in packets:
```

Test traces get longer

Ran 100 test runs 100 steps each: 37 of them diverge

Hard to analyze these traces by hand

LLM hallucinated too much in identifying the root cause



TFTP Test Divergence Analysis

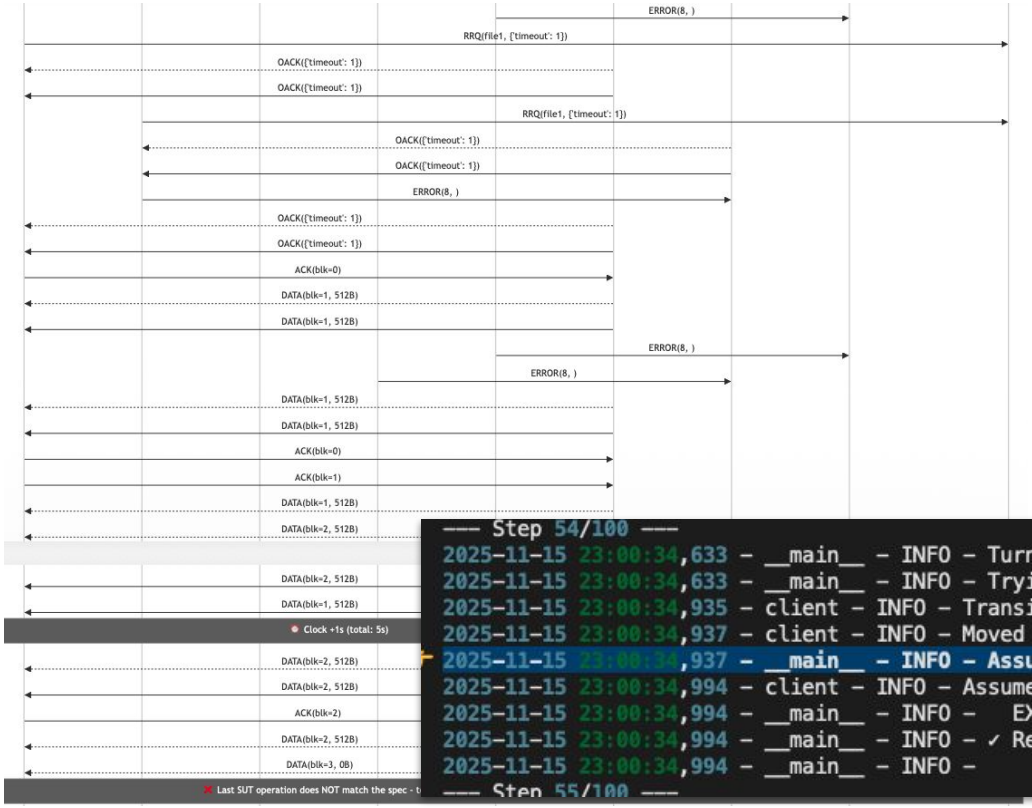
Based on my analysis of all diverging test runs in `test-results/`, here are the key findings:

Summary

37 diverging test runs with **3 main patterns**:

1. **Duplicate DATA packets (26 cases - 81%)**: The SUT sends DATA packets with the correct block number, but these are retransmissions/duplicates
2. **Duplicate OACK packets (5 cases - 16%)**: The SUT sends duplicate OACK packets during negotiation
3. **Unexpected ACK (1 case - 3%)**: Rare case of unexpected ACK packet

Spec vs. Code (7)




```
--- Step 54/100 ---
2025-11-15 23:00:34,633 - __main__ - INFO - Turn: sut. 4 transitions to try
2025-11-15 23:00:34,633 - __main__ - INFO - Trying transition 8 [ServerRecvRRQthenSendData]...
2025-11-15 23:00:34,935 - client - INFO - Transition 8: ENABLED
2025-11-15 23:00:34,937 - client - INFO - Moved to step 55
2025-11-15 23:00:34,937 - __main__ - INFO - Assume lastAction: ActionRecvSend(sent=UdpPacket(srcIp='172.20.0.10',
2025-11-15 23:00:34,994 - client - INFO - AssumeState: ENABLED
2025-11-15 23:00:34,994 - __main__ - INFO -   EXECUTE ACTION: ActionRecvSend(sent=UdpPacket(srcIp='172.20.0.10',
2025-11-15 23:00:34,994 - __main__ - INFO - ✓ Received packet matches the spec
2025-11-15 23:00:34,994 - __main__ - INFO -
--- Step 55/100 ---
```

The server starts data exchange

In the middle, the server spec interprets an earlier RRQ as no-negotiation case, and sends DATA(1) again!

```
diff --git a/spec/tftp.tla b/spec/tftp.tla
index c48f936..df41aaa 100644
--- a/spec/tftp.tla
+++ b/spec/tftp.tla
@@ -145,6 +145,8 @@ _ServerSendDataOnRrq(_rrq, _clientIpAndPort, _newServerI
                transferred |-> dataSize
            ]
        IN
+       \* The no-negotiation case of RFC 2347. No options were requested.
+       /\  DOMAIN _rrq.options = {}
        /\  packets' = packets \union { dataPacket }
        /\  serverTransfers' = [
                p \in DOMAIN serverTransfers \union {_clientIpAndPort} |->
@@ -163,6 +165,8 @@ _ServerSendOackOnRrq(_rrq, _clientIpAndPort, _newServerI
     ServerRecvRRQthenSendOack::
     \E optionsSubset \in SUBSET DOMAIN _rrq.options,
            blksize \in 0..65464, timeout \in 1..255:
+       \* RFC 2349: option negotiation
+       /\  DOMAIN _rrq.options /= {}
        \* RFC 2349, Section 3.1: "If the server is willing to accept
        \* the blocksize option, it sends an Option Acknowledgment
        \* (OACK) to the client.  The specified value must be less
```

FIX #7:

1. Apply RFC 2347 case only when options is empty
2. Apply RFC 2349 case only when options are non-empty

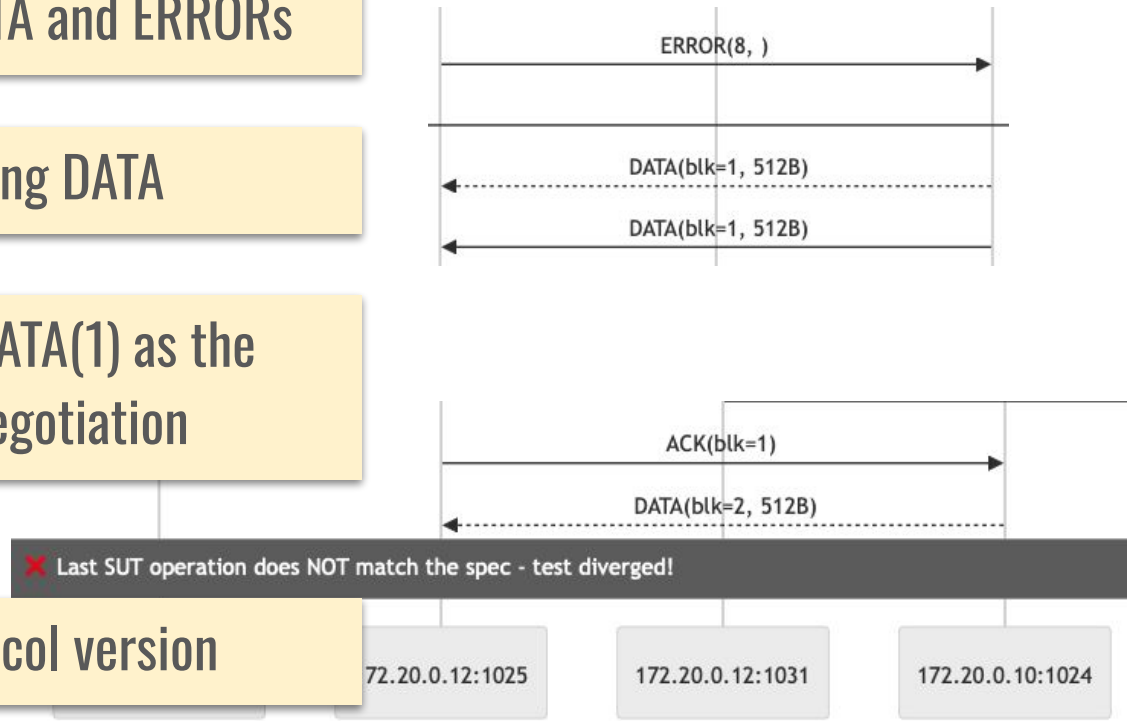
Spec vs. Code (8)

The client receives DATA and ERRORs

The server keeps sending DATA

The client interprets DATA(1) as the new session without negotiation

FIX #8: Save the protocol version



FIX #9: The client must send tsize = 0 in RRQ

FIX #10: The server should send default timeout if it's not specified in the options

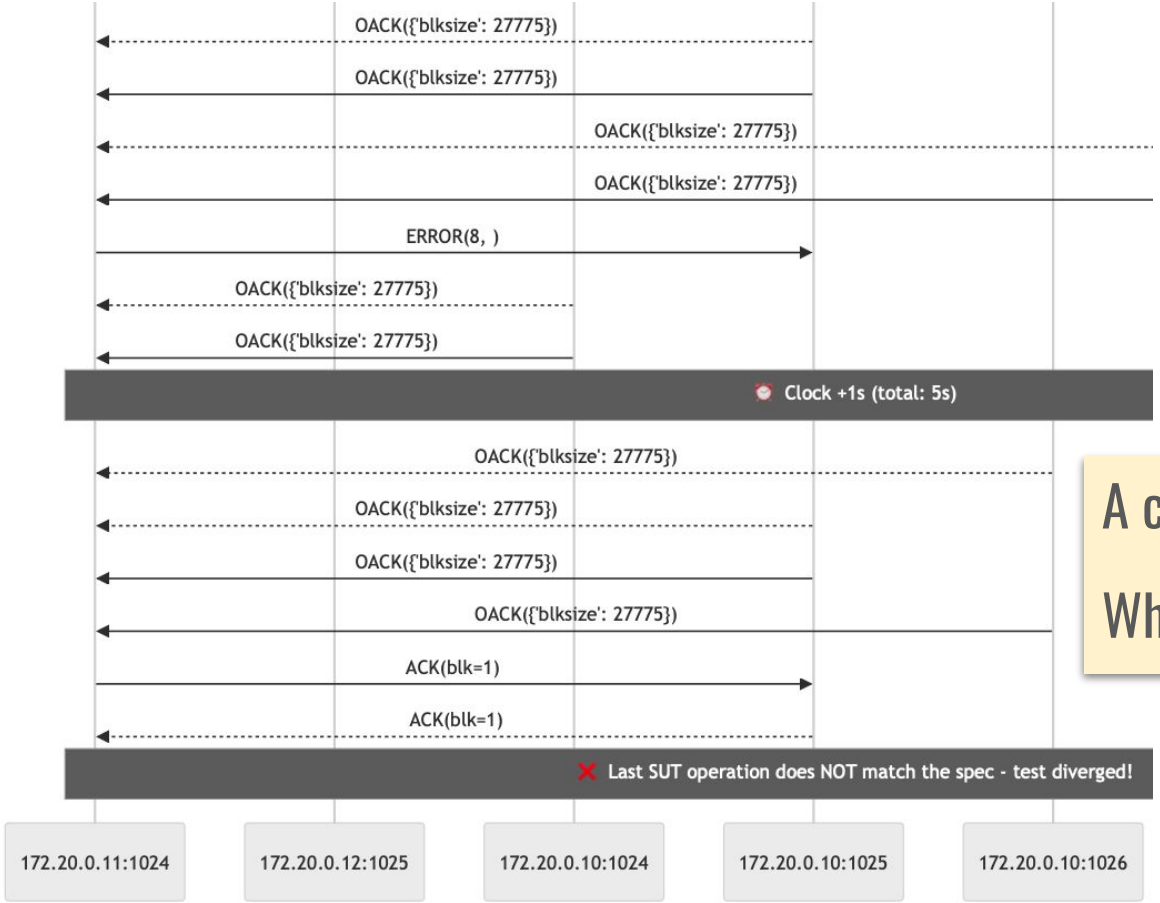
FIX #11: The server may send invalid (e.g., outdated) packets

FIX #12: My understanding of TFTP timeouts was wrong

FIX #13: Handle ERROR packets

FIX #14: Receive only one OACK message

FIX #15: Use clientIP-clientPort-serverPort triplets



A confusing run.

Why is server ACKing on ACK?

Are all of our specifications broken?

The verification engineer's mindset:

The specification usually **overapproximates** the implementation

Reachability: if the impl. reaches a state *s*, then the spec. reaches α(s)

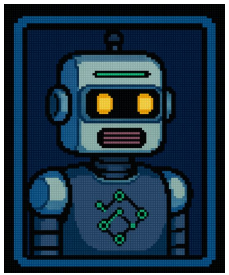
Conformance testing (roughly):

If the spec. executes action *A*, the impl. must be able to execute *A*

If the impl. executes action *A*, the spec. must be able to execute *A*

Plenty of research in the 1990es

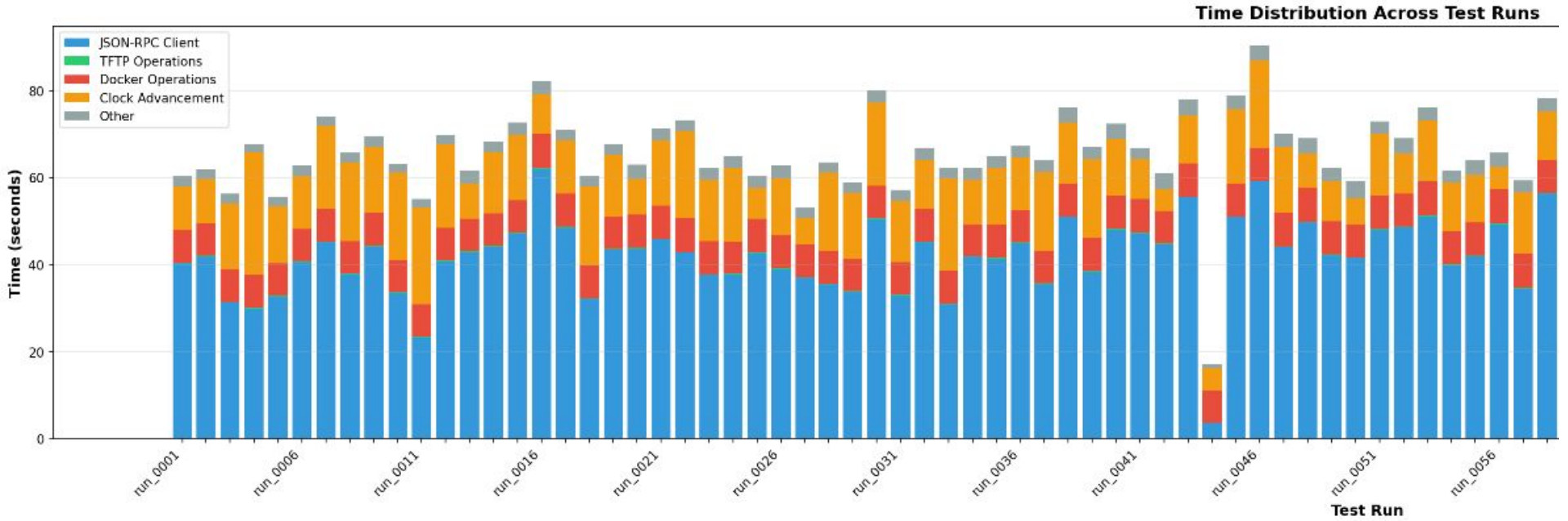
A few lessons from LLM-generated harnesses



1. Surprisingly, **it works** – **not effortlessly** though
2. **Do not trust** the generated harness – TODOs and bugs inside
3. **Do not let** an LLM define **its own formats** – it goes wild
4. Log yourself and **define your log format** – or, face wild regexes
5. **Generate visualizations** that suit your needs – it's **amazing**!

Produce 100 episodes, 100 steps each

```
$ ./harness.py --docker --steps=100 --tests=100
```

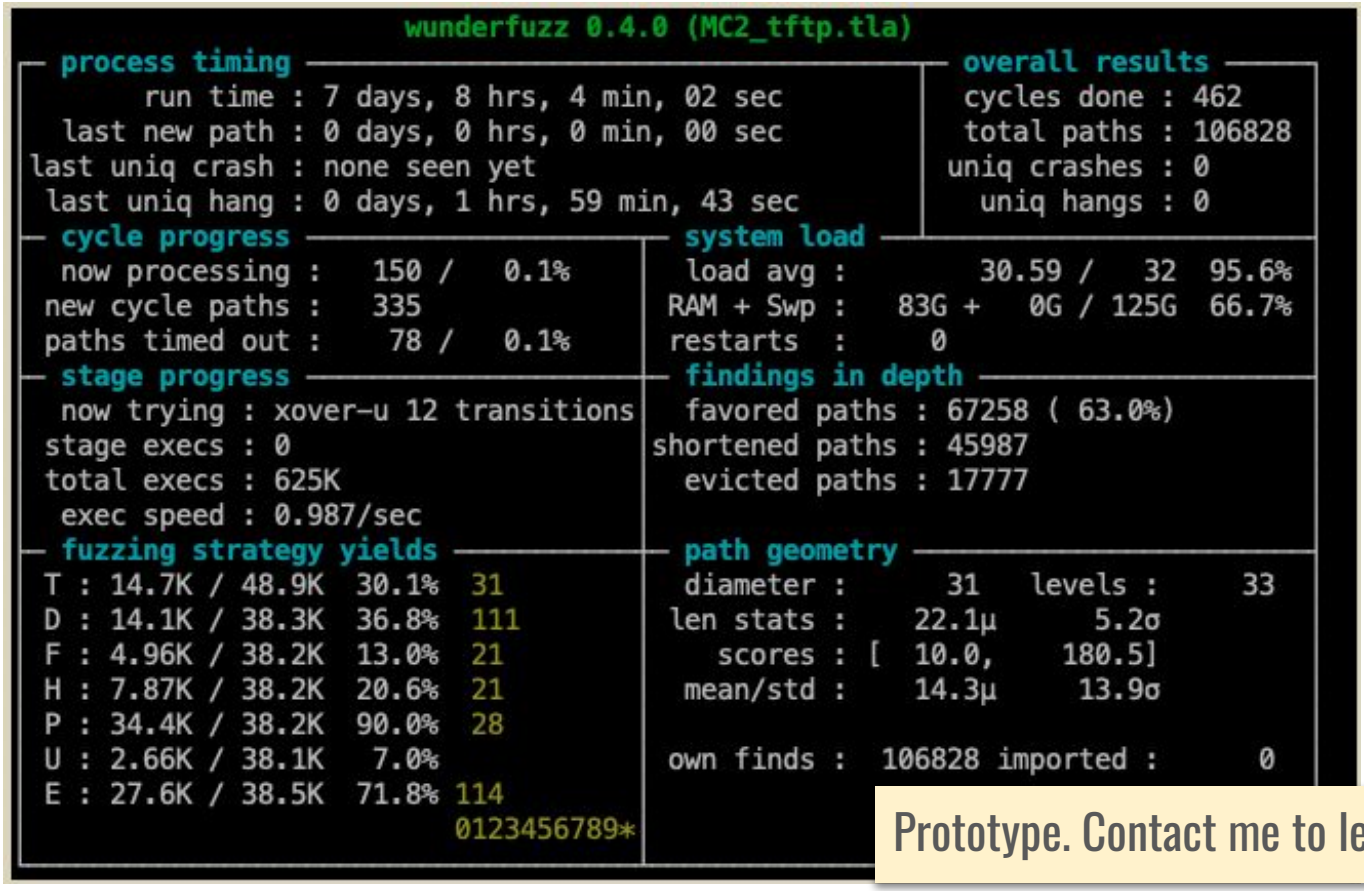


Improvements?

Docker restarts and `time.sleep` are **slow** (seconds)

Deterministic simulation instead?

Guided search = **fuzzing** + **symbolic execution**



Prototype. Contact me to learn more

Testing with TLA+

1. Nagendra et. al. ***Model guided fuzzing of distributed systems*** (2025)
2. Cirstea, Kuppe, Merz, Loillier. *Validating Traces of Distributed Systems Against TLA+ Specifications* (2024)
3. Chamayou et. al. *Validating System Executions with the TLA+ Tools* (2024)
4. Jordan Halterman. *Verifiability Gap: Why We Need More From Our Specs and How We Can Get It* (2020)
5. Jessie Davis et al. *eXtreme Modelling in Practice* (2020)
6. Kupriyanov, Konnov. *Model-based testing with TLA+ and Apalache* (2020)
7. Pressler. *Verifying Software Traces Against a Formal Specification with TLA+ and TLC* (2018)

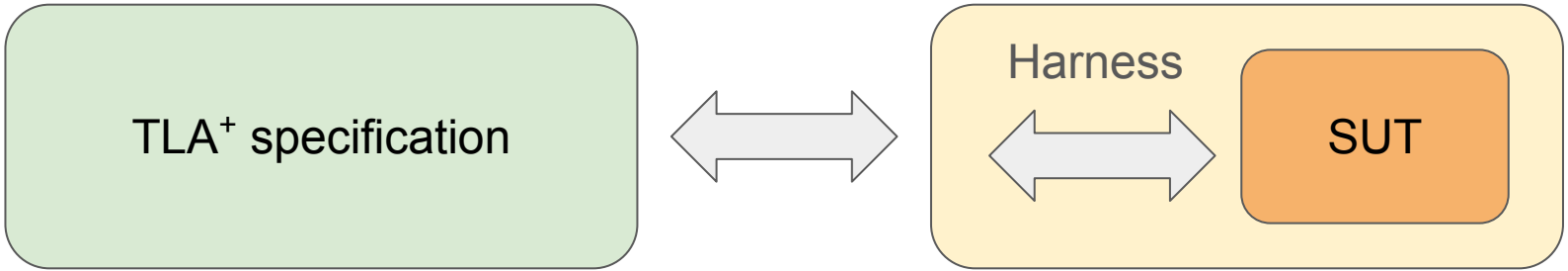
Except [6], all use TLC

Pros of our approach

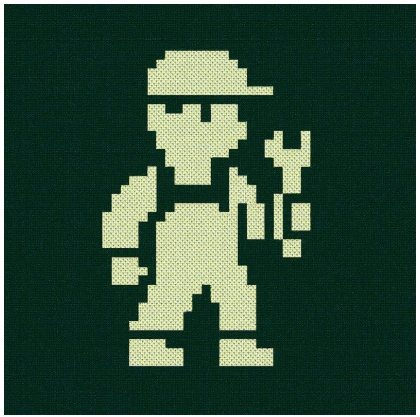
Black-box testing: communication over the network, no instrumentation

Minimal mapping: define the labels of TESTER and SUT

Modular: test components in isolation, the rest is the spec



Takeaways



1. Write your **own symbolic search** scripts!
2. **Connect your spec with the code**, as you like it
3. **LLMs can help** you, when it’s too boring or too diverse
4. You know better **what matters to your project**
5. **Put it in your CI**
6. Easy to **parallelize**

Q & A

The spec and the harness:

[github.com/konnov/tftp-symbolic-testing]

e-mail: igor@konnov.phd telegram: igor_konnov_phd

Select fitter transitions

Filter tests with views

```
\* Compute the measure as the max of block numbers of all transfers.
\* @type: (<<Str, Int>> -> $transfer) => Int;
Measure(t) ==
    LET MaxS(S) == ApaFoldSet(LAMBDA x, y: Max(x, y), 0, S) IN
    MaxS({ t[ipPort].blockNum : ipPort \in DOMAIN t })

\* Use the number of transferred blocks as the measure for computing
\* the fitness function. The rest of the view is used for filtering
\* similar states.
\* @type: << Int, <<Str, Int>> -> $absTransfer, <<Str, Int>> -> $absTransfer >>;
MeasureView == <<
    (1 + Measure(serverTransfers) + Measure(clientTransfers)) * 10,
    [ p \in DOMAIN serverTransfers |-> AbsTransfer(serverTransfers[p]) ],
    [ p \in DOMAIN clientTransfers |-> AbsTransfer(clientTransfers[p]) ]
>>
```

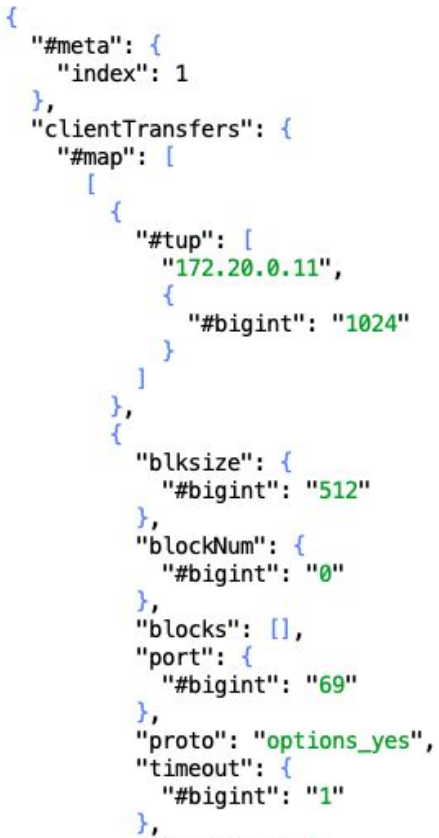
```
AbsTransfer(t) ==
    [ port |-> t.port,
      \* -1, 0, [1, ...)
      tsize |-> Min(t.tsize, 1),
      \* abstract [0, 512), [512, 1024), [1024, 1536), ...
      blksize |-> t.blksize \div 512,
      \* interval abstraction [0, 10], [11, 255]
      timeout |-> IF t.timeout <= 10 THEN 10 ELSE 255,
      \* keep exact blockNum
      blockNum |-> t.blockNum,
      \* simply whether we are below or above the timeout threshold
      timestamp |-> IF clock <= t.timestamp + t.timeout THEN -1 ELSE 1
    ]
```

This is pure TLA+.

No tool extension needed

ITF JSON Traces

[apalache-mc.org/docs/adr/015adr-trace.html]



```
{
  "#meta": {
    "index": 1
  },
  "clientTransfers": {
    "#map": [
      [
        {
          "#tup": [
            "172.20.0.11",
            {
              "#bigint": "1024"
            }
          ]
        },
        {
          "blksize": {
            "#bigint": "512"
          },
          "blockNum": {
            "#bigint": "0"
          },
          "blocks": [],
          "port": {
            "#bigint": "69"
          },
          "proto": "options_yes",
          "timeout": {
            "#bigint": "1"
          },
```

```
trace_json = {
    "#meta": {"id": 23},
    "params": ["N"],
    "vars": ["pc", "x"],
    "loop": 0,
    "states": [
        {
            "#meta": {"no": 0},
            "N": {"#bigint": "3"},
            "pc": "idle",
            "x": {"#bigint": "42"},
        },
        {
            "#meta": {"no": 1},
            "pc": "lock",
            "x": {"#bigint": "43"},
        },
    ],
}
```

Apalache produces ITF traces

Trivial to parse:

see github.com/konnov/itf-py

Type checker

apalache-mc.org/docs/tutorials/snowcat-tutorial.html

Damas & Milner type inference + row types (no inductive types)

Resolving type imprecision between function-like types

May require type annotations for records, tuples, functions, and sequences

Int	Bool	Str
UNINTERPRETED	Set(a)	Seq(a)
a -> b	<<a, b, c>>	{ f1: a, f2: b, f3: c }
(a, b, c) => d		Tag1(a) | Tag2(b) | Tag3(c)

Translation to SMT

TLA⁺ Model Checking Made Symbolic [OOPSLA'19]

Mimic the semantics implemented by TLC – explicit model checker

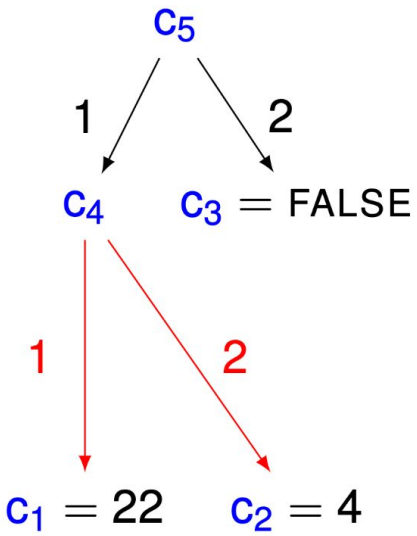
Compute **layout of data structures**, constrain **contents with SMT**

Define operational semantics via reduction rules – for bounded data structures

Trade efficiency for expressivity

Static picture of TLA$^+$ values and relations between them

Arena:



SMT:

integer `sort Int`

Boolean `sort Bool`

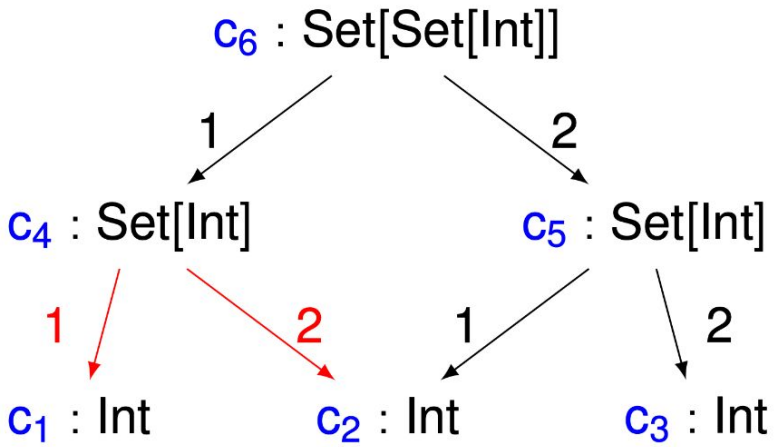
name, e.g., `"abc"`, uninterpreted sort

finite set:

- a constant c of uninterpreted sort set_τ
- propositional constants for members

  $in_{\langle c_1, c \rangle}, \ldots, in_{\langle c_n, c \rangle}$

Arenas for sets: { { 1, 2 }, { 2, 3} }



SMT defines the contents, e.g., to get $\{\{1\},\{2\}\}$:

$$in_{\langle c_1,c_4\rangle} \wedge \neg in_{\langle c_2,c_4\rangle} \wedge in_{\langle c_2,c_5\rangle} \wedge \neg in_{\langle c_3,c_5\rangle}$$

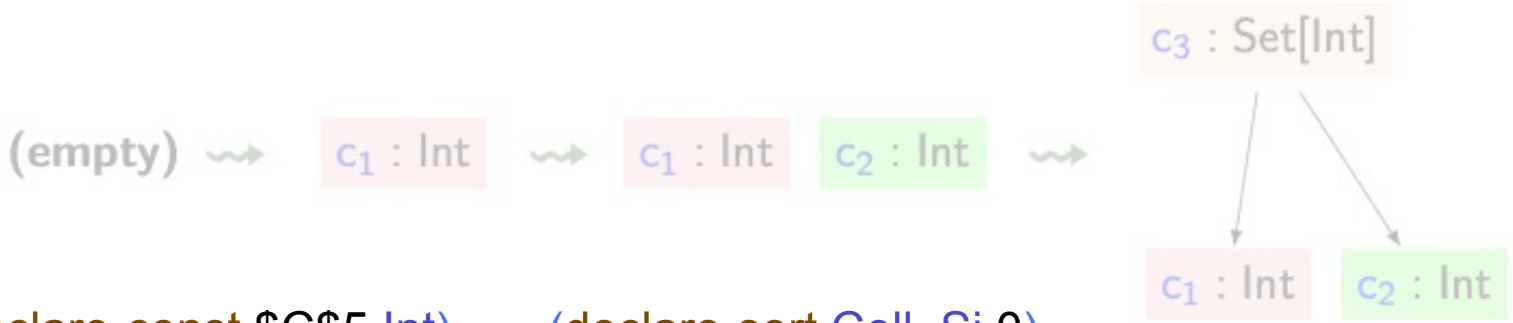
Rewriting the set construction

$\{1, 2\} \rightsquigarrow \{c_1, 2\} \rightsquigarrow \{c_1, c_2\} \rightsquigarrow c_3$

Corresponding arena

Rewriting the set construction

Corresponding arena




```
(declare-const $C$5 Int)
(assert (= $C$5 1))
(declare-const $C$6 Int)
(assert (= $C$6 2))
```

```
(declare-sort Cell_Si 0)
(declare-const $C$7 Cell_Si)
(declare-const in_i5_Si7 Bool)
(declare-const in_i6_Si7 Bool)
(assert in_i5_Si7)
(assert in_i6_Si7)
```