Validating System Executions with the TLA+ Tools

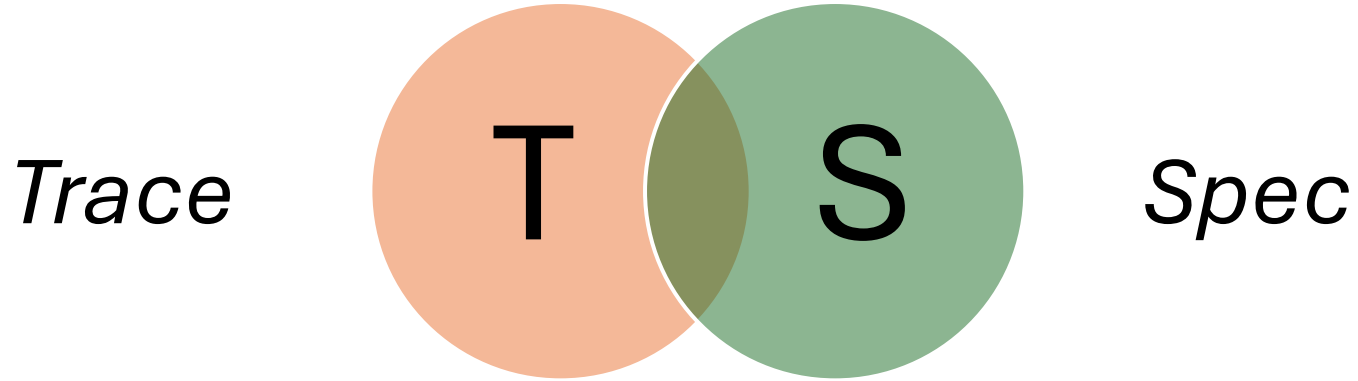
TLA+ Conf 2024

Amaury Chamayou [2], Benjamin Loillier [4], Eddy Ashton [2], Eric Dai [1], Heidi Howard [2], Horatiu Cirstea [4], Jian Zhou [1], Joshua Zhang [1], Markus A. Kuppe [3], Stephan Merz [4], Vincent Li [1]

[1] Microsoft Azure, [2] Microsoft Azure Research, [3] Microsoft Research, [4] University of Lorraine, CNRS, Inria, LORIA, Nancy, France

TLA+ Trace Validation: In a Nutshell

1. Each node locally logs relevant events
2. Merge node-local logs into single, global *Log*
3. Generate set of behaviors T defined by "trace spec" *Trace* that conform to *Log*
4. Check if $T \cap S \neq \emptyset$, where S is the set of behaviors defined by high-level spec *Spec*



Prior Work

- *Using formal specifications to monitor and guide simulation: Verifying the cache coherence engine of the Alpha 21364 microprocessor* (2002, Tasiran et al.)
  - Concurrent system + custom tailored to simulator

- *Verifying Software Traces Against a Formal Specification with TLA+ and TLC* (2018, Ron Pressler)
  - Outlined technique toy example

- *eXtreme Modeling in Practice* (2020, Jessie Davis et al.)
- *Bridging the Verifiability Gap: Why We Need More From Our Specs and How We Can Get It* (2020, Jordan Halterman)

Spec driven development

System	Spec	Implementor	Findings
Produce/Consumer MPMC Queue	Kuppe	Kuppe	Single-Mutex bug (deliberate)
Distributed Termination detection (EWD998)	Kuppe	Kuppe	Token rounds initiated after global termination
Two-Phase Commit protocol	Lamport	Inria	List instead of set to count resource managers (timeouts cause same RM to be counted multiple times)
Consistency of a Key-Value Store	Demirbas	Inria	Snapshot Isolation issue due to implementation not creating a proper snapshot at TX start
Distributed Consensus (Raft)	Ongaro	Inria	Integer division bug (missing ceiling) causing candidate to incorrectly reach quorum

etcd-raft with Azure

Goal and execution: Add novel Raft feature (2 nodes + witness)

- Trace validation vanilla etcd-raft #111
- Raft with witness support #133
- Inefficiency: next index shall be larger than match index #149

serathius commented on Jan 25 Member

+1 to witness support. My main concern would be creation of a test plan to ensure correctness. I think the TLC model checker will be crucial here.

xiang90 commented on Nov 27, 2023 Contributor

I chatted with both Lamport/Yuan in 2015 to discuss this issue (https://www.microsoft.com/en-us/research/publication/specifying-and-verifying-systems-with-tla/)

Here is the "conclusion" from them:

> There were multiple efforts in the past to compile TLA+ spec to actual code, but as far as I can tell none of them were very successful. One promising approach is to write a mapping (i.e., refinement function) that maps implementation states in for example C++ to specification state. This would allow you to check the specification as a property of the implementation. It also allows you to use the specification to drive the testing of the implementation.

Releases 233

v3.5.13 Latest
2 weeks ago

+ 232 releases

Packages

No packages published

Used by 9.6k

Contributors 839

+ 825 contributors

Languages

- Go 96.5%
- Shell 2.0%
- Jsonnet 1.1%
- Other 0.4%

CCF with Azure Research

IA-CCF: Individual Accountability for Permissioned Ledgers

Alex Shamis[1,2], Peter Pietzuch[1,2], Burcu Canakci[+3], Miguel Castro[1], Cédric Fournet[1], Edward Ashton[1], Amaury Chamayou[1], Sylvan Clebsch[1], Antoine Delignat-Lavaud[1], Matthew Kerner[4], Julien Maffre[1], Olga Vrousgou[1], Christoph M. Wintersteiger[1], Manuel Costa[1], and Mark Russinovich[4]

[1]Microsoft Research, [2]Imperial College London, [3]Cornell University, [4]Microsoft Azure

Abstract

Permissioned ledger systems allow a consortium of members that do not trust one another to execute transactions safely on a set of replicas. Such systems typically use Byzantine fault tolerance (BFT) protocols to distribute trust, which only ensures safety when fewer than 1/3 of the replicas misbehave. Providing guarantees beyond this threshold is a challenge: current systems assume that the ledger is corrupt and fail to identify misbehaving replicas or hold the members that oper-

cas misbehave. With more misbehaving replicas, current permissioned ledger systems can no longer be trusted. When safety violations are detected, the whole service is deemed

CCF: A Framework for Building Confidential Verifiable Replicated Services

Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak*, Olga Ohrimenko, Felix Schuster*, Roy Schuster, Alex Shamis, Olga Vrousgou, Christoph M. Wintersteiger

Microsoft Research & Microsoft Azure

April 2019

***Abstract*—We present CCF, a framework to build permissioned confidential blockchains. CCF provides a simple programming model of a highly-available data store and a universally-verifiable log that implements a ledger abstraction. CCF leverages trust in a consortium of governing members and in a network of replicated hardware-protected execution environments to achieve high throughput, low latency, strong integrity and strong confidentiality for application data and code executing on the ledger. CCF embeds consensus protocols with Byzantine and crash fault-tolerant configurations. All configurations support strong service integrity based on the ledger contents. Even if some replicas are corrupt or their keys are compromised, they can be blamed based on their signed evidence of malicious activity recorded in the ledger. CCF supports transparent, programmable governance**

for millions of customers. Some of their designs do not address confidentiality [5], [23] while others provide confidentiality but relatively low performance (e.g., about 4 transactions per second [17]).

In this paper, we present CCF, a framework that addresses these limitations: it provides both confidentiality and high-performance for consortium-based blockchains. CCF replicates its blockchain operations using a network of hardware-protected trusted execution environments (TEEs). This yields high throughput, high availability, and low latency, while at the same time protecting the integrity and confidentiality of application data and code running on the ledger. Although CCF

any TEE, our initial implementation uses Intel are Guard Extensions) enclaves [49]. Enclaves protected memory regions that allow trustworthy code even on untrusted host computers. Some hains designs also use enclaves [47], [71], [72], do not provide confidentiality guarantees.

members of a consortium may not necessarily other, and need only agree on the service they

Confidential Consortium Framework: Secure Multiparty Applications with Confidentiality, Integrity, and High Availability

Heidi Howard* Azure Research, Microsoft

Fritz Alder† imec-DistriNet, KU Leuven Belgium

Edward Ashton Azure Research, Microsoft

Amaury Chamayou Azure Research, Microsoft

Sylvan Clebsch Azure Research, Microsoft

Manuel Costa Azure Research, Microsoft

Antoine Delignat-Lavaud Azure Research, Microsoft

Cédric Fournet Azure Research, Microsoft

Andrew Jeffery† University of Cambridge UK

Matthew Kerner Microsoft

Fotios Kounelis† Imperial College London UK

Markus A. Kuppe Microsoft Research

Julien Maffre Azure Research, Microsoft

Mark Russinovich Microsoft

Christoph M. Wintersteiger Azure Research, Microsoft

ABSTRACT

Confidentiality, integrity protection, and high availability, abbreviated to CIA, are essential properties for trustworthy data systems. The rise of cloud computing and the growing demand for multiparty applications however means that building modern CIA systems is more challenging than ever. In response, we present the Confidential Consortium Framework (CCF), a general-purpose foundation

may wish to keep data confidential to protect intellectual property, for competitive advantage, or to maintain systems security, for instance when storing secrets. Encryption at rest and in-flight are well-established approaches to achieving confidentiality, but confidentiality during execution is more challenging. Moreover, encryption alone does not fully solve the problem of confidentiality. Instead, it reduces the problem of protecting arbitrary data into

Raft-Inspired CFT consensus

- Dynamic reconfiguration
- Cryptographic guarantees
- Foundation of Azure offerings

TLA+ spec written after the fact

Validating traces of ~15 impl tests revealed several spec bugs:

- Empty & Batching of entries in AppendEntries msgs #5150 #5154
- …
- Real-world bootstrapping #5828, node membership #5902, and node retirement #5919
- …
- Propose request vote message to speed up some reconfigurations #5697
- Support modeling different network guarantees #5634

Releases 182

4.0.16 Latest

3 weeks ago

+ 181 releases

Contributors 55

+ 41 contributors

Languages

- C++ 67.4%
- Python 23.8%
- TLA 3.0%
- TypeScript 2.6%
- CMake 1.9%
- Shell 0.7%
- Other 0.6%

CCF with Azure Research (contd.)

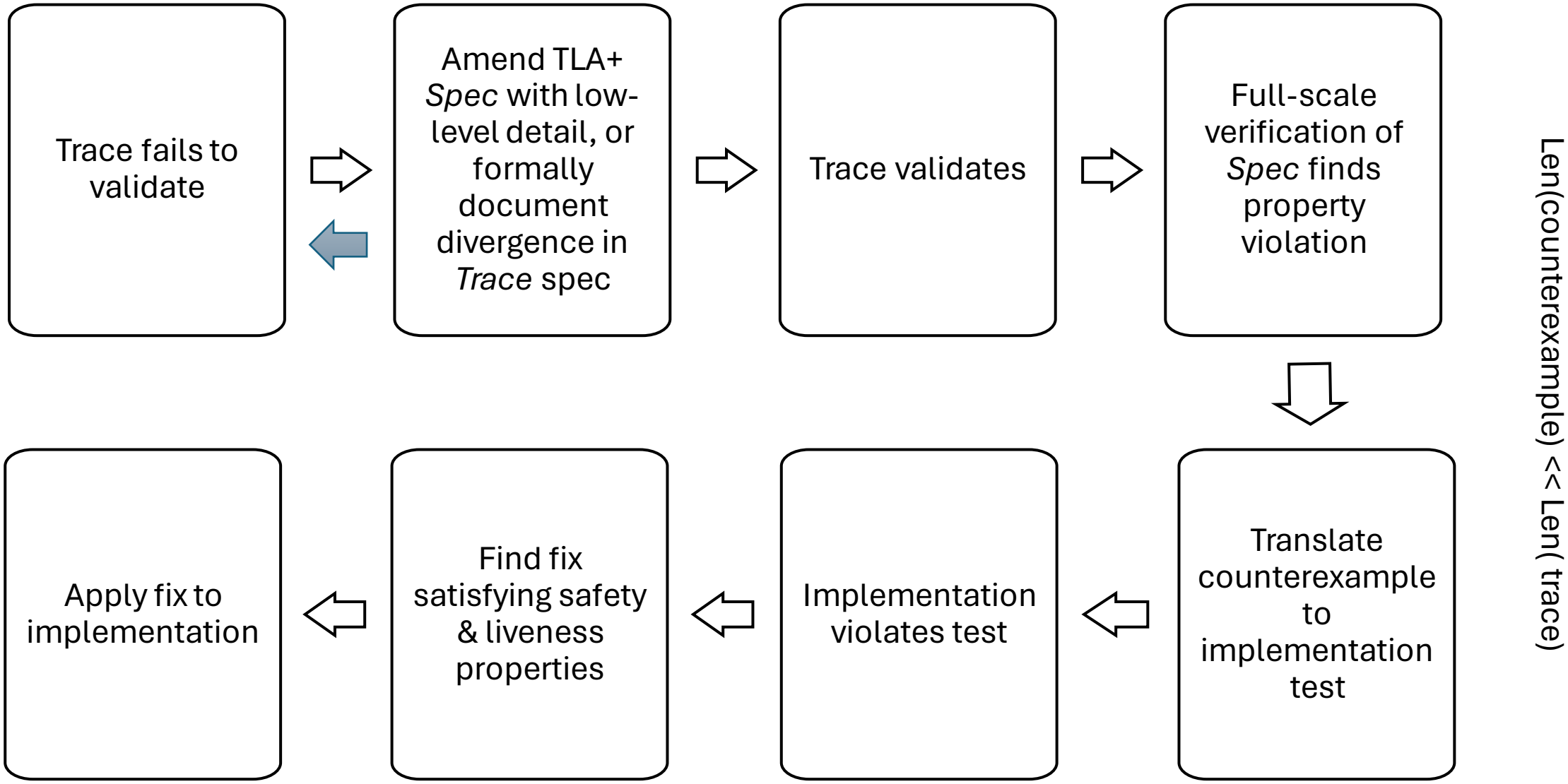
“Safety violation due to reuse of the Term field in Append Entries messages #5927”

- Related to processing of (stale) ACKs and NACKs, i.e., happy and non-happy path
- Found and diagnosed by *system* experts

“[R]euse of match_idx can lead to unsafely advancing commit index #5325”

- Related to processing of NACKs, i.e., non-happy path
- Found and diagnosed by *FM* expert

Why bugs despite passing/green tests?



TV mini tutorial - EWD998

https://github.com/tlaplus/Examples/pull/75/

Spec (EWD998Chan.tla)

```
Next ==

   \* Some computation with async messaging
   \/ \E n \in Node :
        SendMsg(n) \/ RecvMsg(n) \/ Deactivate(n)

   \* Termination detection with sync messaging
   \/ InitiateToken \* node 0
   \/ \E n \in Node \ {0} : PassToken(n)
```

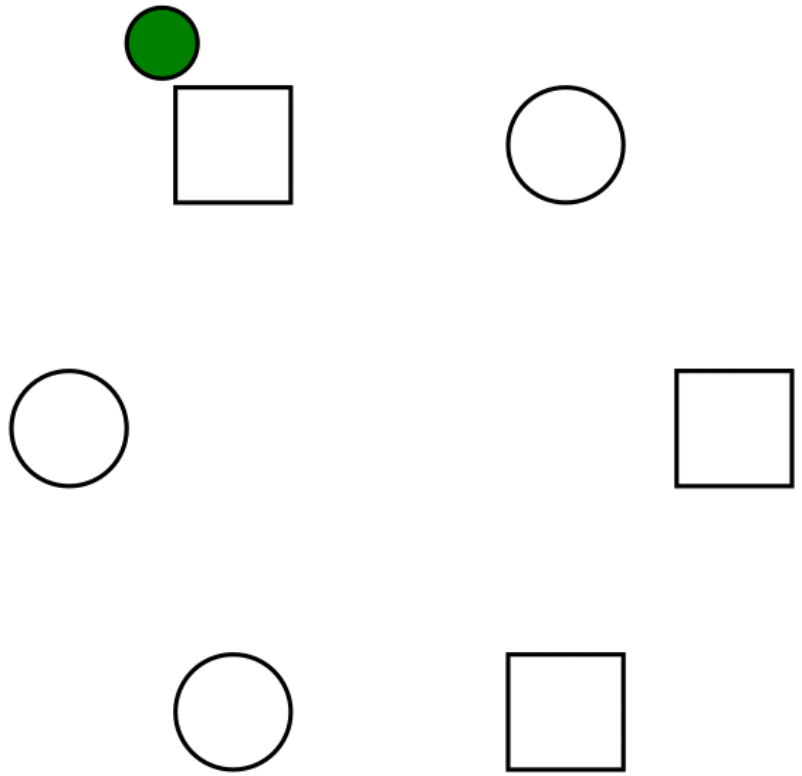
EWD998: Termination Detection

Circle: Active, Black: Tainted

Line: Message, Arrow: Receiver

Dashed: In-Flight, Solid: Arrival in next

Level: 1 Terminated: F Detected: F

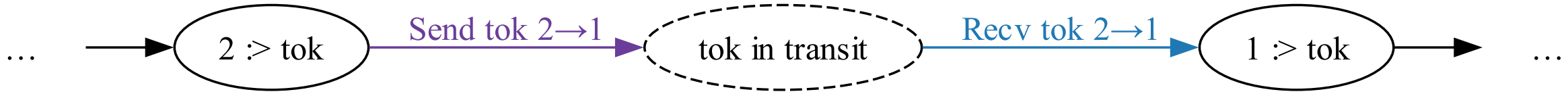


Implementation: EWD998.java

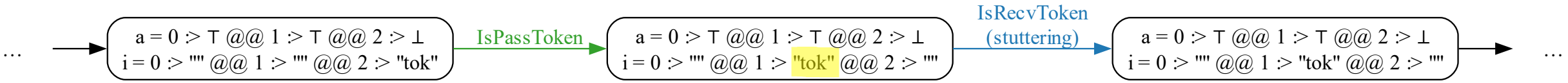
```
void sendMsg(int sender, int receiver, Object msg) {

   JsonObject pkt = new JsonObject();
   pkt.add(SND, sender);
   pkt.add(RCV, receiver);
   pkt.add(MSG, msg);
   pkt.add(VC, clock.tick());

   [...]
   socket.send(pkt);

   JsonObject logline = new JsonObject();
   logline.add(EVENT, ">");
   logline.add(NODE, sender);
   logline.add(PKT, pkt);
   System.out.println(logline);
```

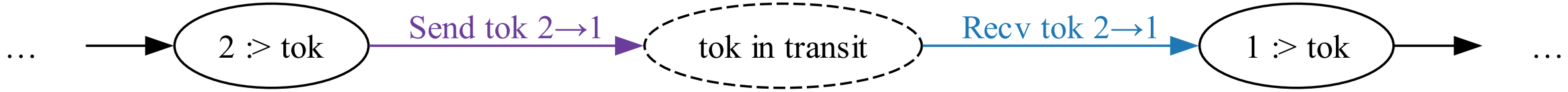
Trace (EWD998ChanTrace.tla)

```
EXTENDS EWD998Chan, Json, VectorClock

VARIABLE length

Log == CausalOrder(Deserialize("log.ndjson"), …)
line == Log[length]

IsSendMsg ==
 /\ length \in DOMAIN Log /\ length’ = length + 1
 /\ line.event = ">” /\ line.pkt.msg.type = "pl"
 /\ <<SendMsg(line.pkt.snd)>>_vars
 \* Receiver non-deterministic in SendMsg.
 /\ IsPrefix(inbox[line.pkt.rcv], inbox’[line.pkt.rcv])
```

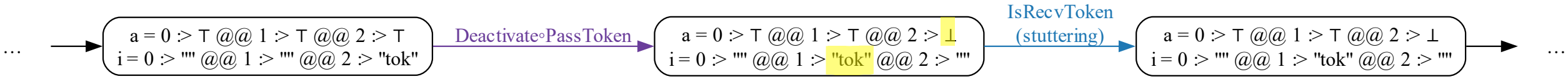


Add explicit stuttering action:





Compose *Deactivate* and *PassToken* actions:



Verification: |T|>1 (non-determinism)

Cannot check:	Reason:
Safety	No state violates anything
Deadlock	Spurious counterexamples
Liveness: <>[]*length* = *Len*(*Log*)	Spurious counterexamples
EF length = *Len*(Log)	Not expressible in TLA (LTL)

Can check:	
Liveness: *[](length <= Len(Log) => []TLCGet("queue") > 0 …)*	Kludge, but some candidate behavior ☺
Post condition: *TLCGet("stats").diameter = Len(Log)*	True/False by default, but *-dump dot,actionlabels,colorize,constrained,snapshots trace.dot*

Logging: Best Practices

Log when:

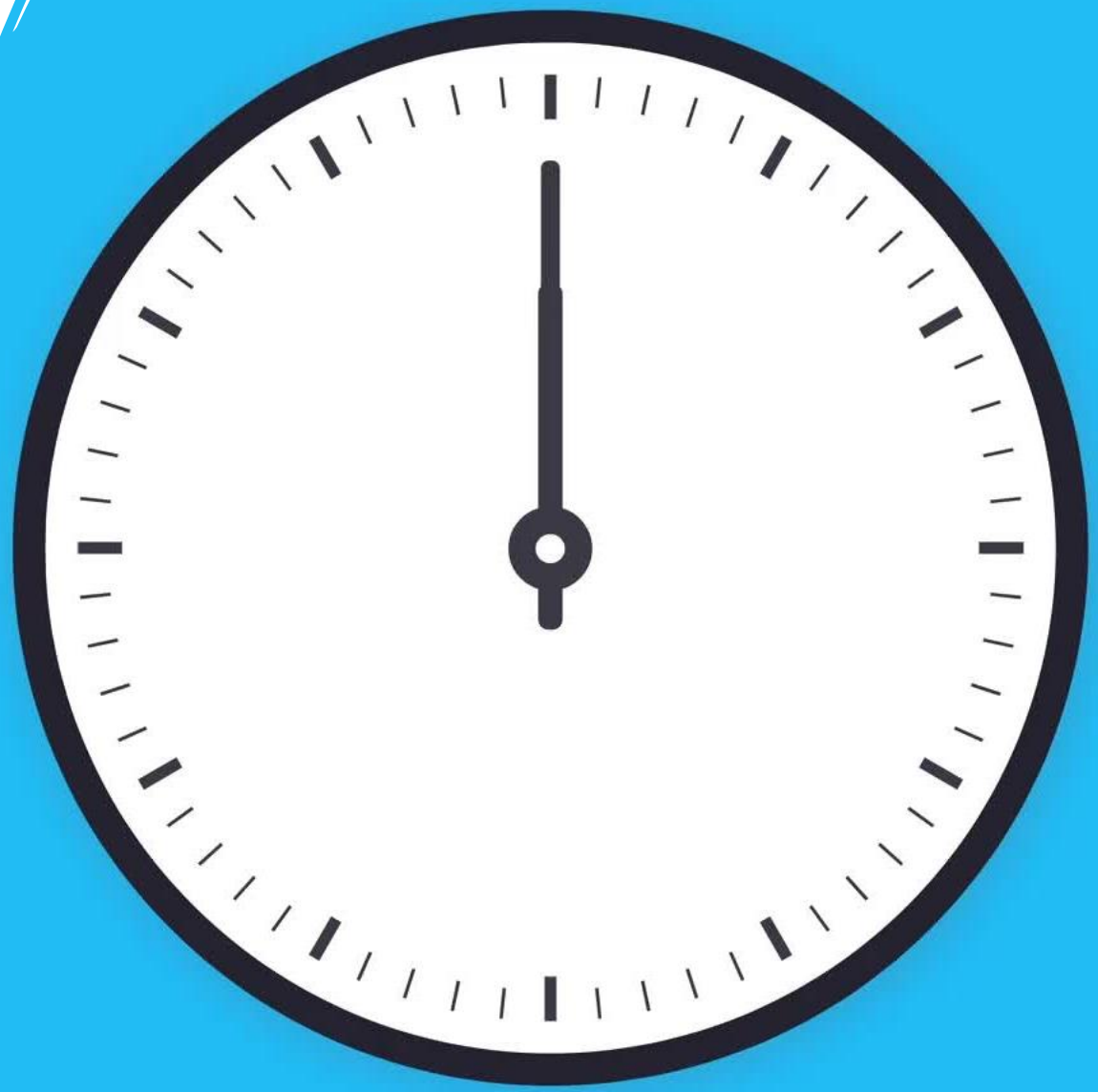
- Messages are sent and received
- Node-local, observable state changes
- Include primed and unprimed values
- O(1) space variable values

etcd (11 log statements total)	CCF (15 log stmts total)
Send & Rcv of *AppendEntries, RequestVote*, …	Send & Rcv…
State changes to *Leader, Follower, Candidate*	State changes…
Configuration changes (*Add*)	Configuration changes…
Advance Commit Index	Advance Commit Index
	Test infra drops messages (reduce non-determinism)

Logging: Causality

- Centralized Clock if you must
- Distributed Clock if you can
  - Code changes or space might be prohibitive
  - TLA+ CommunityModules: Vector Clock
    - Code taken from ShiViz ☺

Bonus VC: Interactive time-space diagrams



Conclusion

- TLA+ tools mature to narrow the spec to code gap

- TV found spec <> impl divergences in all 7 systems
- …identified non-trivial bugs in real-world systems
- …helps reverse-engineer impl into spec
- …helps specs and impls stay in sync
  - Even if non-TLA+ engineers change impl

- TV requires TLA+ expertise
  - Engineers involved in etcd and CCF effort know TLA+
    - https://github.com/microsoft/CCF/pull/6119

[Next]_v

- Does TV generalize?!
  - => How detailed does a spec have to (Raft spec is very detailed)?
  - => Small-scope hypothesis vs at scale?
    - TLC's (new) DFS mitigates SSE iff $T \cap S \neq \emptyset$

- How to generate a *diverse* set of traces?
  - Fuzzing, Chaos engineering, … guided by spec coverage

- Model-based testing (generate behaviors and have impl replay)
  - How to trigger faults/failures?

Questions?