Lupin COPES with blockchain consensus

Francisco Santos[1] Leonardo Santos[1] Marco Giunti[2]
António Ravara[3,4] Simão Melo de Sousa[3,5]

Inforum
September 6, 2024

[1]University of Beira Interior [2]University of Oxford [3]NOVA LINCS

[4]NOVA School of Science and Technology [5]University of Algarve

Consensus Protocols are challenging

Moderately Complex Paxos Made Simple:
High-Level Executable Specification of Distributed Algorithms*

Yanhong A. Liu Saksham Chand Scott D. Stoller

Computer Science Department, Stony Brook University

Comparing Distributed Consensus Algorithms*

Péter Urbán
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa 923-1292, Japan

André Schiper
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Paxos vs Raft: Have we reached consensus on distributed consensus?

Heidi Howard
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

Richard Mortier
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

Changes to (blockchain) protocols may and have led to vulnerabilities

Ethereum moving to Proof of Stake became open to bouncing attacks

Ethereum proof-of-stake under scrutiny.

U. Pavloff, Y. Amoussou-Guenou, and S. Tucci-Piergiovanni 2022

Solana halting has been proven to vulnerable to halt with a single malicious node

Halting the Solana Blockchain with Epsilon stake.

J. Sliwinski et al. 2024

The distance from theory to practice

```
function propose(c)
  if ∄s : ⟨s, c⟩ ∈ decisions then
    s′ := min{s | s ∈ ℕ+ ∧
        ∄c′ : ⟨s, c′⟩ ∈ proposals ∪ decisions};
    proposals := proposals ∪ {⟨s′, c⟩};
    ∀λ ∈ leaders : send(λ, ⟨propose, s′, c⟩);
  end if
end function
```

Paxos Proposer pseudocode

Paxos Made Moderately Complex, Robbert van Renesse 2019

```
proctype proposer(int round; short myval) {
    short hr = -1, hval = -1, tmp;
    short h, r, v;
    byte count;
    bprepare(round);
    do
     :: rec_p(round,count,h,v,hr,hval);
     :: send_a(round,count,hval,myval,tmp);
     od }
```

Paxos Proposer in SPIN

Model Checking Paxos in Spin, Giorgio Delzanno et. All 2014

Typically, consensus protocols are:

- described with pseudo code
- verified with idealised languages
- implemented with mainstream languages

COnsensus Protocols Evaluation and Simulation (COPES)

Aim

- Support evaluation and validation by simulated executions
- Detect/check vulnerabilities by running extensively (many experiments, run many times)

Pipeline for developing and analysing prototypes of consensus protocols

1. Specify in a declarative event-based DSL with well defined semantics
2. Analyse the specification
3. Compile (in proved correct-by-construction way) to a mainstream language
4. Run the code on a simulator / emulator

COnsensus Protocols Evaluation and Simulation (COPES)

Aim

- Support evaluation and validation by simulated executions
- Detect/check vulnerabilities by running extensively (many experiments, run many times)

Pipeline for developing and analysing prototypes of consensus protocols

1. Specify in a declarative event-based DSL with well defined semantics
2. Analyse the specification
3. Compile (in proved correct-by-construction way) to a mainstream language
4. Run the code on a simulator / emulator

Lupin

- Domain Specific Language (*DSL*) to develop consensus mechanisms
- Foundations written in formal language (Coq)
- Implementation guided by formal specification
- Generates OCaml code runnable in consensus simulators (E.g. MOBS COPES)

Example: formal semantics of receive block

$$(\textsc{RcvValidB})\ \frac{\begin{array}{c} E = \{\mathsf{id} = i,\ \mathsf{received\_blocks} = br,\ \mathsf{chain} = bc,\ \ldots\} \\ c = \mathsf{neighbours}\ i \quad m = \mathsf{Block}\{b, \ldots\} \\ m * c \in M \quad m \in br \quad \mathsf{valid(b)} \\ E' = E\ \{\mathsf{chain} = bc + m,\ \mathsf{minting} = \mathit{true},\ \ldots\} \end{array}}{M\ @\ E\ \triangleright\ N_i \xrightarrow{c\ ?\ m} M\ @\ E'\ \triangleright\ N_i}$$

- M is the mailbox
- E is the entity
- N_i is a role with identity i
- c is a set of channels abstracting the neighbours of i

Lupin implementation of rule

```
| MINER + receive Block from neighbours as msg =>
  if contains msg entity.chain do
    if valid msg do
        entity.chain := add msg entity.chain;
        entity.minting := true;
    end
  end
```

Semantic analyser

- Type checking, verification of conditions in simulation
- Protocol simulated in functor-based Virtual Machine
- Each node evaluated at each execution step
- Ending on consensus or error (rules never executed), discover potential deadlocks and livelocks

Example: building blocks on alarm rings

```
File .lue
id : int
blockchain : list block

File .lum
Block {
  id : int
  hash : string
}

File .lun
Entities = 1..128
Topology =
  PartiallyConnectedUndirected

File .lup
protocol = Test
roles = MINER

RUN {
| MINER + spontaneously =>
    if entity.id = 1 do
      Alarm.start 2;
    end
  + alarm_ring =>
    if entity.id = 1 do
      Alarm.stop ();
      a := Digest.SHA.create_hash
        (Random.string 128);
      msg := Block (entity.id) a;
      entity.blockchain := List.append
        entity.blockchain msg;
      send msg to neighbours;
    end
  + receive Block from neighbours
    as msg =>
    if not List.contains
    entity.blockchain msg do
      entity.blockchain := List.append
        entity.blockchain msg;
      send msg to neighbours;
    end }
```

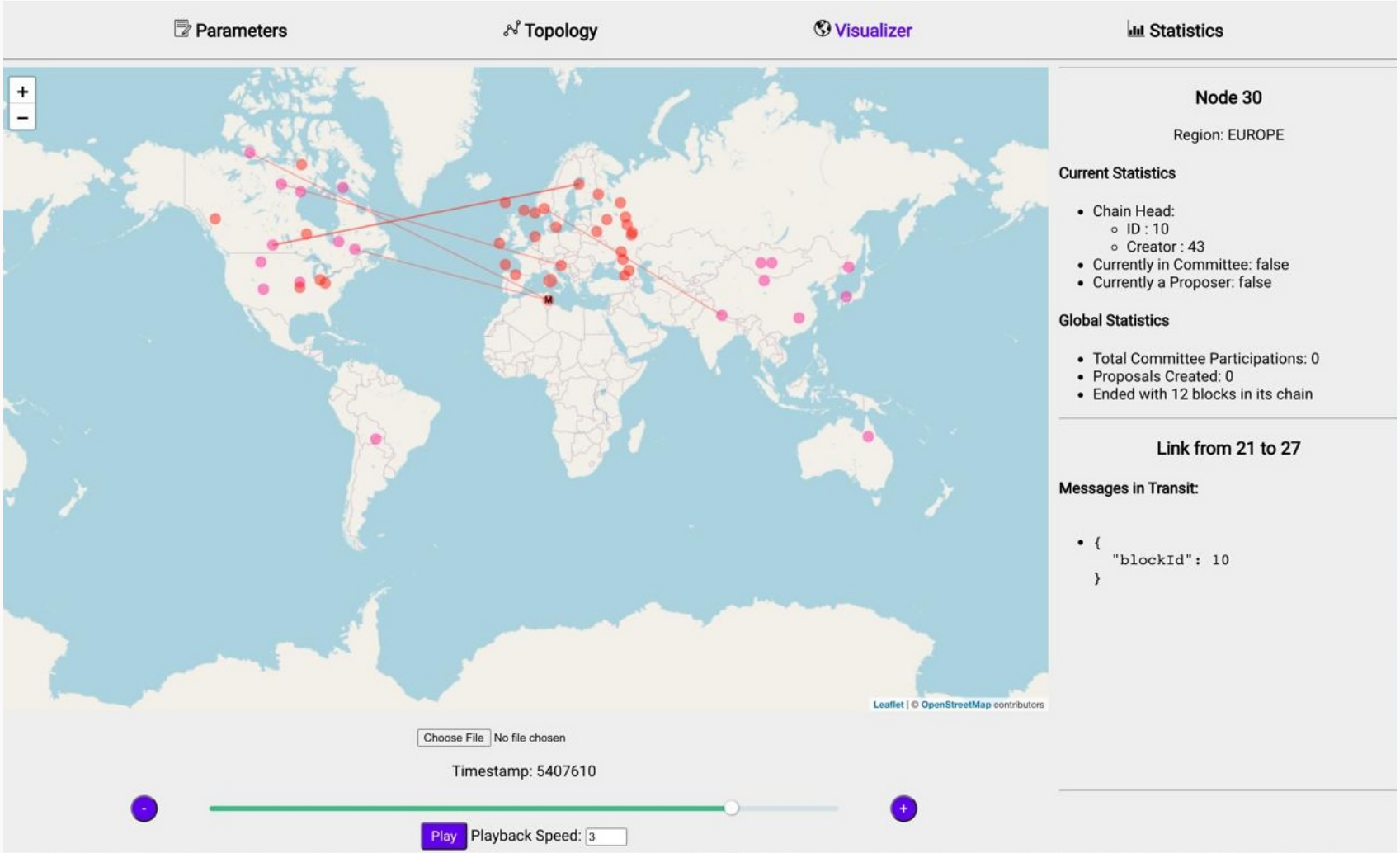
Generated OCaml code

4-tuple:

1. type of rule
2. configuration to be checked
3. message constructor
4. function executed if rule accepted

```
    ( Lts Receive,
    Conf (entity, !role, peer),
    Some (Constructor "Block"),
    fun ()→
      let m = match in_msg with Some msg→ msg | None→ assert false in
      let content =
        match m with
        | Make (_, _, [ _; String content ])→ content
        | _→ assert false
      in
      let blockchain =
        get_field entity "blockchain" |> fun a→
        match a with Some (Messages lst)→ lst | _→ assert false
      in
      if List.mem m blockchain then ()
      else (
        UnorderedSet.add out_buffer (Some m);
        entity.f1 <- Some ("blockchain", Messages (m :: blockchain)))
  );
```

Visual simulation of protocol



Discussion

This talk:

- Developed DSL tailored for protocol engineers
- Programmer focuses on protocol logic
- Consensus mechanism deployed by Lupin

Future work:

- Mechanise properties of underlying model
- Add support for simulators and checkers, e.g. PRISM

Anyone interested?

Thanks!