Kaizen: Building a Performant Blockchain System Verified for Consensus and Integrity

Faria Kalim†, Karl Palmskog*, Jayasi Mehar‡, Adithya Murali†, P. Madhusudan† and Indranil Gupta†

†University of Illinois at Urbana-Champaign
*KTH; work done while at UT Austin and UIUC
‡Facebook; work done while at UIUC
Blockchains and Cryptocurrencies

Diagram:
- Ethereum
- Bitcoin
  - transactions + state
  - dissemination
- distributed ledger
  - consensus
- Algorand
- Nakamoto
- Ouroboros
Consensus Protocol Challenges

Distributed protocols need to handle:
- communication delays (asynchrony)
- node crashes, corruption
- message drops, duplication, forging

Protocol implementation challenges:
- conformance to protocol specification
- node-local performance
- absence of bugs compromising safety
## Consensus System Formal Verification

<table>
<thead>
<tr>
<th>Project</th>
<th>Paper</th>
<th>Protocol</th>
<th>Tool</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disel</td>
<td>POPL ’18</td>
<td>2-phase commit</td>
<td>Coq</td>
<td>5k+</td>
</tr>
<tr>
<td>Verdi Raft</td>
<td>CPP ’16</td>
<td>Raft</td>
<td>Coq</td>
<td>50k+</td>
</tr>
<tr>
<td>Velisarios</td>
<td>ESOP ’18</td>
<td>PBFT</td>
<td>Coq</td>
<td>50k+</td>
</tr>
<tr>
<td>Ironfleet</td>
<td>SOSP ’15</td>
<td>Paxos</td>
<td>Dafny</td>
<td>20k+</td>
</tr>
<tr>
<td>Toychain</td>
<td>CPP ’18</td>
<td>proof-of-X</td>
<td>Coq</td>
<td>10k+</td>
</tr>
</tbody>
</table>
### Interactive vs. Mostly-Automated Verification

#### Coq proof assistant
- much training required
+ explicit proofs
+ many libraries
- purely functional (extraction to OCaml/Haskell)

#### Dafny verification environment
+ less training required
- implicit proofs
- few libraries
+ functional & imperative (C# code generation)
Our Contributions

- novel combination of Coq & Dafny to build performant and verified blockchain system, Kaizen
- methodology based on continuous refinement
  - adapted & instantiated Coq model
  - translated Coq code (not proofs) to Dafny imperative code
  - refined C# code and linked to network shim
- performed evaluation measuring minting and consensus time
Methodology Overview*

PHASE 1

Abstract protocol design & verification in Coq

STAGE I

Coq experts

PHASE 2

Refinement in Coq

STAGE II

Dafny experts

PHASE 3

Translation of abstract protocol to Dafny contracts

PHASE 4

Refinement to imperative code in Dafny

PHASE 5

Refinement in Dafny for performance

PHASE 6

Implementing application specific functions in C#

PHASE 7

Translation to executable code on a distributed network

Systems engineers

Dafny experts and systems engineers

*system is fully verified until Stage III
1. encode system in higher-order functional language (Gallina)
2. prove specification **interactively** using powerful **tactics**
3. check soundness of every low-level step
Record Block := mkB { prevBlockHash : Hash;
  txs : seq Transaction; proof : VProof }.

Record State := mkS { id: Address; peers: seq Address;
  forest: map Hash Block; txpool: seq Transaction }.

Definition valid_chain_block (bc:seq Block) (b:Block):=
  VAF (proof b) bc (txs b) && all [pred t | txValid t bc] (txs b).
Toychain Results and Extensions

- Toychain proves that in quiescent state, all nodes know the same (canonical) chain
- We added support for coinbase transactions
- We added checking of proof-of-work validity of chains
- We changed Toychain nodes to avoid unnecessary messages

All changes are proof-preserving and now merged into Toychain.
Stage II: Refinement and Verification Using Dafny

1. encode programs and their contracts in imperative language
2. try to prove automatically that contracts are fulfilled
3. add more annotations if necessary

Diagram:

- User
- Dafny
- IVL program
- Boogie
- VCs
- Z3
datatype Block = Block(prevBlockHash: Hash, txs: seq<Transaction>, proof: VProof)

datatype State = Node(id: Address, peers: seq<Address>, forest: map<Hash, Block>, txpool: seq<Transaction>)

class StateImpl {
  var id : Address;
  var peers : ...; var forest : ...; var txpool : ...;
  ghost var st: State;

  predicate Valid() { ...}

  method ProcMsgImpl(from: Address, msg: Message, ts: Timestamp)
    returns (pt: seq<Packet>)
  requires Valid();
  ensures Valid();
  ensures st = procMsg(old(st), from, msg, ts).0;
  ensures pt = procMsg(old(st), from, msg, ts).1;
  { ...}
}
Stage III: Refinements in C#

- block and proof-of-work generation
- define and inject miner rewards
- store pre-computed chains
- add network shim based on UDP
Implementation Architecture

Verified Implementation

- Bitcoin Functions
- Blockchain Functions
- Message Queue

Shim Layer

Network Peer

Verified Implementation

- Bitcoin Functions
- Blockchain Functions
- Message Queue

Shim Layer

Network Peer

...
Evaluation Setup

- metrics: block minting time and consensus time
- use 30-node cluster of 2.4GHz processors w/ 64GB RAM
- baseline: performance of stock Bitcoin implementation
- workload: traces of arrival times of 50 transactions from realistic dataset
Evaluation: Block Minting Time

![Graph showing the time to mint a block for different numbers of blocks and node counts for Bitcoin and KznCoin.](image-url)
Evaluation: Consensus Time

- Initial Number of Blocks: 0, 1, 2
- Time to Consensus (Seconds): 100, 500, 1000
- Bitcoin
- KznCoin

Graph showing the time to consensus for different initial numbers of blocks (100, 500, 1000) for Bitcoin and KznCoin.
Evaluation: Scalability

Cluster Size

Time to Consensus (Seconds)

Bitcoin - 50
KznCoin - 50
Bitcoin - 250
KznCoin - 250
## Components and Effort

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coq refinement</td>
<td>≈ 1k</td>
</tr>
<tr>
<td>Dafny refinement</td>
<td>≈ 5k</td>
</tr>
<tr>
<td>C# refinement</td>
<td>≈ 1k</td>
</tr>
<tr>
<td>C# network shim</td>
<td>≈ 4k</td>
</tr>
</tbody>
</table>

Development effort ≈ 6 person months across four people
Lessons Learned and Future Work

- holistic expertise necessary in Coq/Dafny/systems for Kaizen
- “easy” change can require large changes at earlier stages
- local node computation took most effort to optimize (rather than network messaging)
- future Coq proofs of Toychain Byzantine tolerance transferrable to Kaizen (see WIP by Gopinathan and Sergey, CoqPL ’19)
Conclusion

- System development methodology combines interactive and mostly-automated verification, **Coq & Dafny**
- Verified executable blockchain system **Kaizen**
- Evaluation gives encouraging results on performance

More information:

- GitHub: https://github.com/palmskog/kaizen
- Contact me: **Karl Palmskog** palmskog@kth.se