OmniLedger: A Secure, Scale-Out, Decentralized Ledger

Eleftherios Kokoris Kogias†, Philipp Jovanovic†, Linus Gasser†, Nicolas Gailly†, Ewa Syta*, Bryan Ford†

†EPFL, Switzerland
*Trinity College, USA

Workshop on Blockchain Technology and Theory
2017-10-16, Vienna
Talk Outline

• Motivation
• OmniLedger
• Evaluation
• Conclusion
Talk Outline

- **Motivation**
- OmniLedger
- Evaluation
- Conclusion
Drawbacks of Nakamoto Consensus

• Transaction confirmation delay
  ‣ Any transaction takes *at least* 10 mins until being confirmed

• Weak consistency
  ‣ You are not really certain your transaction is committed until you wait 1 hour or more

• Low throughput
  ‣ Bitcoin: ~7 tx/sec

• Proof-of-work mining
  ‣ Wastes huge amount of energy
Scaling Bitcoin is Not Easy

ONE DOES NOT SIMPLY SCALE BITCOIN
Talk Outline

- Motivation
- Omniledger
- Evaluation
- Conclusion
OmniLedger – Design Goals

**Security Goals**

1. **Full Decentralization**
   No trusted third parties or single points of failure

2. **Shard Robustness**
   Shards process TXs correctly and continuously

3. **Secure Transactions**
   TXs commit atomically or abort eventually

**Performance Goals**

4. **Scale-out**
   Throughput increases linearly in the number of active validators

5. **Low Storage**
   Validators do not need to store the entire shard TX history

6. **Low Latency**
   TX are confirmed quickly

Assumptions: \(\leq 25\%\) mildly adaptive Byzantine adversary, (partially) synchronous network, UTXO model
1. **Sharding**: Periodically re-assign validators to shards in a randomized manner (using RandHound)

2. **Consensus**: Validators ensure consistency of shard states (using Omnicon)

3. **TX Commit**: Clients ensure consistency of cross-shard transactions (using Atomix)
Sharding

1. Temp. leader election (VRF-based)
2. Randomness generation (RandHound)
3. Shard assignment (using $r_{nd_e}$)

Goal:
- Prevent (adaptive) adversary from subverting an entire shard with high probability

Solution:
- Periodically re-assign validators to shards using unbiasable, publicly-verifiable randomness
**Consensus**

**Goal:**
- Ensure shard state consistency (process TX, etc.)

**Solution:** Omnicon
- Variant of ByzCoin
- Group- instead of tree-based communication
  ‣ Trade-off some scalability for higher fault tolerance
  ‣ Performs better for practically relevant configurations
- BlockDAG instead of blockchain
  ‣ Capture dependencies between TXs
  ‣ Better performance due to better resource utilization
Transaction Commit

Goal:
- Cross-shard TX commit atomically or abort eventually

Solution: Atomix
- Client-managed protocol
  1. Client sends cross-shard TX to input shards
  2. Collect acceptance/rejection proofs from input shards
  3. (a) If all input shards accepted, commit to output shard, otherwise (b) abort and reclaim input funds
- Optimistically trust client for liveness
  ‣ Anyone can take over the clients job if he times out
- Collective signing (CoSi) ensures compact proofs

The Atomix protocol for secure cross-shard transactions
Trust-but-Verify Transaction Validation

Goal:

- Avoid latency vs. throughput trade-off

Solution:

- Use two-level “trust-but-verify” validation
  - Low latency:
    - Optimistically validate transactions batched into small blocks (e.g., 500KB)
  - High throughput:
    - Batch optimistically validated blocks into bigger blocks (e.g., 16MB) and re-validate
Talk Outline

• Motivation
• OmniLedger
• Evaluation
• Conclusion
Implementation & Experimental Setup

**Implementation**

- Go versions of OmniLedger and its subprotocols (Omnicon, Atomix, etc.)
- Based on DEDIS code
  - Kyber crypto library
  - Network library
  - Cothority framework
- [https://github.com/dedis](https://github.com/dedis)

**DeterLab Setup**

- 48 physical machines
  - Intel Xeon E5-2420 v2 (6 cores @ 2.2 GHz)
  - 24 GB RAM
  - 10 Gbps network link
- Network restrictions
  - 20 Mbps bandwidth
  - 200 ms round-trip latency
Evaluation: Throughput

Results for 1800 validators

Scale-out throughput for 12.5%-adversary and shard size 70
## Evaluation: Latency

Transaction confirmation latency in seconds for regular and multi-level validation

<table>
<thead>
<tr>
<th>#shards, adversary</th>
<th>4, 1%</th>
<th>25, 5%</th>
<th>70, 12.5%</th>
<th>600, 25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular validation</td>
<td>1.38</td>
<td>5.99</td>
<td>8.04</td>
<td>14.52</td>
</tr>
<tr>
<td>1st lvl. validation</td>
<td>1.38</td>
<td>1.38</td>
<td>1.38</td>
<td>4.48</td>
</tr>
<tr>
<td>2nd lvl. validation</td>
<td>1.38</td>
<td>55.89</td>
<td>41.89</td>
<td>62.96</td>
</tr>
</tbody>
</table>

Latency increase since optimistically validated blocks are batched into larger blocks for final validation to get better throughput.
Talk Outline

• Motivation
• OmniLedger
• Experimental Results
• Conclusion
Conclusion

• OmniLedger:
  ‣ Secure scale-out distributed ledger framework
  ‣ Sharding through publicly-verifiable unbiasable randomness (via RandHound)
  ‣ Intra-shard BFT consensus (via Omnicon)
  ‣ Client-managed cross-shard TX (via Atomix)
  ‣ Avoids latency vs. throughput tradeoff (via trust-but-verify TX validation)
  ‣ Visa-level throughput and beyond

• Full paper: ia.cr/2017/406
Thank you!

Questions?

Philipp Jovanovic – philipp.jovanovic@epfl.ch – @Daeinar