Hybrid Fault-Tolerant Consensus in Asynchronous and Wireless Embedded Systems

22nd International Conference on Principles of Distributed Systems

Wenbo Xu, Signe Rüsch, Bijun Li, Rüdiger Kapitza

TU Braunschweig

18.12.2018, Hong Kong
Background: (Binary) Byzantine-Fault Tolerant Consensus

- Fundamental problem in distributed systems
- Totally $n$ node in the group, each proposes a value, 0 or 1
- In the end all nodes should decide the same value $\rightarrow$ consensus
Background: (Binary) Byzantine-Fault Tolerant Consensus

- Fundamental problem in distributed systems
- Totally $n$ node in the group, each proposes a value, 0 or 1
- In the end all nodes should decide the same value → consensus
- At most $f$ faulty nodes
  - Crash
  - Byzantine fault: actively work against the algorithm
Background: Asynchronous System

- Nodes communicate via messages
- Asynchronous network
  - No message omissions
  - But messages can take arbitrarily long time
  → Too slow? Or he didn’t send? Cannot wait forever!

- Strong adversary: the worst case
  - The adversary can inspect the status of every message and node
  - … then reorder arrivals of messages, and adjust faulty nodes’ behavior
  - Cannot break cryptography and a trusted subsystem
Background: Hybrid Fault Model

- Trusted subsystem, tamperproof
- A strict monotonic counter to prevent “two-faced cheating”
- Faulty nodes cannot send contradictory messages in one broadcast
Related Work and Motivation

• Randomization to bypass FLP impossibility of asynchrony
  – Crash fault tolerance with $n \geq 2f+1$: Ben-Or’s algorithm [1]
  – Byzantine fault tolerance requires $n \geq 3f+1$

• Limit the Byzantine behavior with a trusted subsystem
  – Only requires $n \geq 2f+1$
  – Built upon complex algorithm stacks, e.g. reliable broadcast primitive
  – Not resilient against strong adversary → not terminate in worst case

2f+1 consensus, but less complex and suitable in wireless embedded systems

Correctness proof under all cases, even strong adversary

Outline

- Trusted-Ben-Or Algorithm
- A Common Issue in the Proof of Termination
- Experiment
Original Ben-Or’s Algorithm

Round based, 2 phases per round

Propose a value 0 or 1

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>VO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ben-Or’s Algorithm

Round based, 2 phases per round

PR: Propose Phase
VO: Vote Phase

Wait for \((n-f)\) proposals

If >\(n/2\) propose the same \(v\)
  → Vote for \(v\)

Else
  → Vote for \(\bot\) (default)

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>VO 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Ben-Or’s Algorithm**

Round based, 2 phases per round

- **PR: Propose Phase**
- **VO: Vote Phase**

Wait for \((n-f)\) votes

If all vote for \(\perp\)

\(\rightarrow\) Propose \((\$, R\),

\$ is a random value

If someone votes for \(v\)

\(\rightarrow\) Propose \((v, D)\)

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VO</td>
<td>0</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>2</td>
<td>PR 0, D</td>
<td>0, D</td>
<td>0, R</td>
</tr>
<tr>
<td>VO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(R = \text{Randomly get the value}\)

\(D = \text{Deterministically get the value}\)
Ben-Or’s Algorithm

Round based, 2 phases per round

- PR: Propose Phase
- VO: Vote Phase

... If \( >n/2 \) vote for the same \( v \) → Decide \( v \)

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>VO 0</td>
<td>( \perp )</td>
<td>( \perp )</td>
</tr>
<tr>
<td>2</td>
<td>PR 0, D</td>
<td>0, D</td>
<td>0, R</td>
</tr>
<tr>
<td></td>
<td>VO 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Ben-Or’s Algorithm is a round-based algorithm with 2 phases per round. If more than \( n/2 \) nodes vote for the same value \( v \), the algorithm decides on \( v \).
Ben-Or’s Algorithm

Round based, 2 phases per round

PR: Propose Phase
VO: Vote Phase
PR: Propose Phase
VO: Vote Phase

Only tolerate crash fault, no Byzantine fault!

### Ben-Or’s Algorithm

#### Round based, 2 phases per round

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>VO 0</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>2</td>
<td>PR 0, D</td>
<td>0, D</td>
<td>0, R</td>
</tr>
<tr>
<td></td>
<td>VO 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Round 1

- **PR (Propose Phase)**: Node 1 proposes 0, Node 2 proposes 1, Node 3 proposes 0.
- **VO (Vote Phase)**: Node 1 votes for 0, Node 2 votes for 1, Node 3 votes for 0.

### Round 2

- **PR (Propose Phase)**: Node 1 proposes 0, Node 2 proposes 0, Node 3 proposes R.
- **VO (Vote Phase)**: Node 1 decides for 0, Node 2 decides for 0, Node 3 decides for 0.

Only tolerate crash fault, no Byzantine fault!
Trusted-Ben-Or: Tackle Byzantine faults

• Message uniqueness per phase
  → Trusted monotonic counter for message authentication

• Unbiased random number
  → Trusted random number generator (combined with the counter)

• Semantic correctness
  → Message certificate
In round $k$, each node only sends 2 messages

- Trusted monotonic counter authentication:
  - $<\text{PR}, k, *, *>$ with counter value $[k|0]$
  - $<\text{VO}, k, *>$ with counter value $[k|1]$

- Trusted random number generator
- Protected by hardware, can only crash but not Byzantine

\[
\text{AUTH}(\text{message}|\text{id}|c^{\text{new}}) \rightarrow ($) + \text{AUTH}(\text{message}|\text{id}|c^{\text{new}}|$)
\]
- Piggyback received, authenticated messages to proof the correctness
- No recursive certificates
  - Limited message size ($\leq n+2$ messages in one certificate)
  - Faulty node can include invalid into a certificate
Adaption to Embedded Wireless Systems

• Local broadcast instead of peer-to-peer communication
• Tackle (limited) omission faults:
  – Stubborn re-transmission of last message
  – Round jumping when received a valid message of future round
    → No specific network protocols / primitives required for reliable communication
• HMAC in trusted subsystem instead of digital signature
Outline

- Trusted-Ben-Or Algorithm
- A Common Issue in the Proof of Termination
- Experiment
Proof of Termination

• No valid proposals of (0, D) and (1, D) at the same time
  
  \[
  \text{PR} \quad \left[ \frac{(n+1)}{2} \times 0 \right] \quad \left[ \frac{(n+1)}{2} \times 1 \right]
  
  \text{VO} \quad (0, D) \quad (1, D)
  
  \]

• In a lucky round:
  – All trusted coins of each node toss the same random value \( v \)
  – … which is the same as the valid deterministic value

→ Terminate in this round
Proof of Termination

A corner case of flaw
• Firstly let a node R-get v

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>0,D</td>
<td>1,D</td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>0,R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proof of Termination

A corner case of possible flaw
• Firstly let a node $R$-get $v$
• Then let another node $D$-get $(1-v)$

$\rightarrow$ Turn the lucky value into unlucky

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>0,D</td>
<td>1,D</td>
<td>1,D</td>
</tr>
<tr>
<td>VO</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>1</td>
</tr>
<tr>
<td>PR</td>
<td>0,R</td>
<td></td>
<td>1,D</td>
</tr>
<tr>
<td>VO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proof of Termination

A corner case of possible flaw

• Firstly let a node R-get v
• Then let another node D-get (1-v)

→ Turn the lucky value into unlucky

<table>
<thead>
<tr>
<th>Round</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>0,D</td>
<td>1,D</td>
<td>1,D</td>
</tr>
<tr>
<td>VO</td>
<td>↓</td>
<td>↓</td>
<td>1</td>
</tr>
<tr>
<td>PR</td>
<td>0,R</td>
<td></td>
<td>1,D</td>
</tr>
<tr>
<td>VO</td>
<td>↓</td>
<td></td>
<td>↓</td>
</tr>
<tr>
<td>PR</td>
<td>1,R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Luckiness” should not depend on future events!

Proof of Termination

• In our work, termination is ensured by:
  – Counter authentication
  – Trusted random number generator
  – Semantic certificate
  – “Luckiness”

• Luckiness depends only on the current system state and past events!

• For more details please refer to our paper
Outline

- Trusted-Ben-Or Algorithm

- A Common Issue in the Proof of Termination

- Experiment
Experiments: Settings

- 3-10 Raspberry Pi 3: ARM processor with TrustZone interface, distributed in different rooms in office building
- Wireless ad-hoc mode, UDP multicast
  - ICMP ping delay: (min, average, max) = (5.6 ms, 12.5 ms, >1000 ms)
  - iperf3 test: up to 24% data loss
- Trusted counter implemented on Linaro OPTEE
  - SHA-256 HMAC provided by OPTEE
- Compare with Turquois [1]

Experiment: Result with Byzantine Faults Injected

- Comparable median
- Higher variance
- Can tolerate more faults

![Graph showing latency results for different numbers of nodes. The graph compares Trusted BenOr and Turquois algorithms.]
Conclusion

• Randomized binary consensus in asynchronous system

• Trusted monotonic counter for message authentication

• Resilient against strong adversary

• Tailored for embedded wireless systems

• Tolerate more faults with limited overhead (in most cases)

Thank you for your attention!
Motivation: Distributed Consensus
Trusted BenOr Algorithm: Overview

1 //the initial round. Round number is omitted
2 send <PR, v, D-get>
3 wait for \( \lceil \frac{n+1}{2} \rceil \) valid PR-messages
4 if \( \lceil \frac{n+1}{2} \rceil \) <P, v> with the same v
5      send <VO, v>
6 else
7      send <VO, ⊥>
8 wait for \( \lceil \frac{n+1}{2} \rceil \) valid VO-messages
9 if \( \lceil \frac{n+1}{2} \rceil \) <VO, v>
10      decide v
11 //new round
12 if at least one <VO, v>
13      send <PR, V, D-get>
14 else send <PR, $, R-get>
## Message Validity: Legality Certificate

<table>
<thead>
<tr>
<th>Message type</th>
<th>When to send</th>
<th>Required certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PR, k+1, v, R-get&gt;</td>
<td>[(n+1)/2] &lt;VO, k, ⊥&gt;</td>
<td>[(n+1)/2] &lt;VO, k, ⊥&gt;</td>
</tr>
<tr>
<td>&lt;PR, k+1, v, D-get&gt;</td>
<td>[(n+1)/2] &lt;VO, k, *&gt; with at least one &lt;VO, k, v&gt;</td>
<td>[(n+1)/2] &lt;PR, k, v, *&gt;</td>
</tr>
</tbody>
</table>
| <VO, k+1, v> | [(n+1)/2] <PR, k+1, v, *> | [(n+1)/2] <PR, k+1, v, *> 
If there is a <PR, k+1, v, D-get>, then add [(n+1)/2] <PR, k+1, v, *> |
| <VO, k+1, ⊥> | [(n+1)/2] <PR, k+1, *, *> with different values | [(n+1)/2] <PR, k+1, *, *> with different values 
Plus [(n+1)/2] <VO, k, ⊥> |
Proof of Agreement

A correct node decides $v$ in round $k$

Exist $[(n+1)/2]$ valid $<VO, k, v>$

Exist $[(n+1)/2] <PR, k, v, *>$

No $[(n+1)/2] <VO, k, \perp>$

No valid $<PR, k+1, 1-v, D-get>$

No valid $<PR, k+1, *, R-get>$

<table>
<thead>
<tr>
<th>Message type</th>
<th>Required certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;PR, k+1, v, D-get&gt;$</td>
<td>$[(n+1)/2] &lt;PR, k, v, *&gt;$</td>
</tr>
</tbody>
</table>
| $<VO, k+1, v>$ | $[(n+1)/2] <PR, k+1, v, *>$ ...

<table>
<thead>
<tr>
<th>Message type</th>
<th>Required certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;PR, k+1, v, R-get&gt;$</td>
<td>$[(n+1)/2] &lt;VO, k, \perp&gt;$</td>
</tr>
</tbody>
</table>
Proof of Agreement

A correct node decides $v$ in round $k$

Exist $\left\lfloor \frac{n+1}{2} \right\rfloor$ valid $<VO, k, v>$$

Exist $\left\lfloor \frac{n+1}{2} \right\rfloor$ <PR, k, v, *>

No valid <PR, k+1, 1-v, D-get>

Only valid <PR, k+1, v, D-get>

No valid <PR, k+1, *, R-get>

No $\left\lfloor \frac{n+1}{2} \right\rfloor$ <VO, k, ⊥>
Proof of Termination

Correct definition of “luckiness”

- **1 is lucky** in round \((3k+1)\)
- In round \((3k-1)\) and \((3k)\), depends on \(t_0\):
  - 0 is lucky before \(t_0\)
  - Since \(t_0\)
    Case A: a majority proposed 0 in \((3k-1)\) → 0 is lucky
    Case B: no majority proposed 0 in \((3k-1)\) → 1 is lucky

“Luckiness” now only depends on current state and past events.

<table>
<thead>
<tr>
<th></th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3k-1</td>
<td>PR</td>
<td>VO</td>
<td></td>
</tr>
<tr>
<td>3k</td>
<td>PR</td>
<td>VO</td>
<td>★</td>
</tr>
<tr>
<td>3k+1</td>
<td>PR</td>
<td>1,R</td>
<td>1,R</td>
</tr>
</tbody>
</table>

\(t_0 = \) the first time a correct node tosses a coin
Trusted Subsystem: BiTrinc

- Restrict Byzantine nodes, not too faulty → Hybrid fault model
  - Most part can still be Byzantine
  - A small trusted part is never Byzantine (crash-fault-only)
- Can be protected by hardware. Example: ARM TrustZone, Intel SGX, other dedicated hardware security modules
- Minimal Trusted Computing Base
  - As simple and small as possible
  - Only critical functions and data