Byzantine Agreement with Less Communication

Idit Keidar, Technion
Shout Out

Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP.
Shir Cohen, Idit Keidar, and Alexander Spiegelman

All You Need is DAG.
Idit Keidar, Oded Naor, Lefteris Kokoris-Kogias, and Alexander Spiegelman
Byzantine Agreement (BA)

- Consensus among $n$ processes
- Up to $f$ can be controlled by an adversary and act arbitrarily

- Byzantine Atomic Broadcast (BAB)
  - Agree on a sequence of messages
  - Reliable Broadcast + Total Order

- A building block for State Machine Replication (SMR)
New Frontiers for BA & BAB

- Permissioned blockchains – shared ledger
- Other FinTech infrastructures
BA Has Been Around for Four Decades

• 2500+, 7000+ citations, resp.
• Traditional use-cases – a handful of processes

Will it scale?
Traditional BFT According to James Mickens

Figure 1: Typical Figure 2 from Byzantine fault paper: Our network protocol
Scalability Challenges

• Communication (word) complexity (of all processes together)
  • $\Omega(n^2)$ lower bound
    In the worst-case, in deterministic algorithms, regardless of synchrony
    [Dolev and Reischuk 1985]
  • Randomization can help

• Synchrony vs. asynchrony
  • Synchrony is not robust, latency bounds defined in minutes
    $\Rightarrow$ Consider asynchrony
  • Randomization required [Fisher, Lynch, Paterson 1985]
Making It Scale

• Solve BA with high probability (WHP) (probability of being correct tends to 1 as $n \to \infty$)
• Complexity: $\tilde{O}(n)$

VRFs

Perfect coin

• Solve BAB with deterministic safety, probabilistic liveness
• Amortized complexity: $O(n)$ or $O(n \log n)$ per agreement

8
Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP

Shir Cohen, Idit Keidar, Alexander Spiegelman

Technion Novi Research

DISC 2020
Contribution

The first sub-quadratic asynchronous BA WHP algorithm
- $\tilde{O}(n)$ word complexity and $O(1)$ expected time
- Safety and Liveness properties are guaranteed WHP
- Binary BA

- Previous sub-quadratic works made synchrony assumptions
  [King and Saia 2011], Algorand [Gilad et al. 2017]
Model

• Asynchronous
• $n$ processes (permissioned)
• Up to $f$ Byzantine processes for $n \approx 4.5f$
• Trusted PKI
  • Inherent for sub-quadratic algorithms
    [Abraham et al. 2019] [Blum et al. 2020] [Rambaud 2020]
• Delayed adaptive adversary:
  • Can use the contents of a message $m$ sent by a correct process for scheduling a message $m'$ only if $m \rightarrow m'$
Verifiable Random Function (VRF)

• A pseudorandom function that provides a proof of its correct computation

• For a secret key $sk$ with a matching public key $pk$
  • $VRF_{sk}(x)$ is a random value
  • Verifiable using $pk$
Use VRFs for

1. Flipping an (imperfect) shared coin
   • First step: $O(n^2)$ word complexity

2. Committee sampling
   • Cryptographic sortition
   • Reduces word complexity to $O(n \log n)$

Following Algorand [Gilad et al. 2017]
Shared Coin with Success Rate $\rho$

All correct processes output $b$ with probability at least $\rho$, for any value $b \in \{0,1\}$
Shared Randomness

- random number + proof
- random number + proof
- random number + proof
- random number + proof
Background: A Simple VRF-Based Shared Coin

- Synchronous [Micali 2017]
- If the minimum VRF is of a correct process, all agree
  - With probability $\geq \frac{2}{3}$
Background: A Simple VRF-Based Shared Coin

- Synchronous
  [Micali 2017]
- If the minimum VRF is of a correct process, all agree
  with probability $\geq \frac{2}{3}$

Requires synchrony
Asynchronous Shared Coin – Take 1

wait for n-f messages
Asynchronous Shared Coin – Take 1

wait for n-f messages, send minimum

wait for n-f messages

return LSB of minimum value
Asynchronous Shared Coin - Analysis

• We prove:
  • $\Omega(\epsilon)$ bound the number of common values
  • our adversary “commits” to them in advance

⇒ With a constant probability, the global minimum is common
Asynchronous Shared Coin - Analysis

• We prove:
  • $\Omega(\epsilon)$ bound the number of common values
  • our adversary “commits” to them in advance

$\Rightarrow$ With a constant probability, the global minimum is common
Use VRFs for

1. Flipping a shared coin
   • First step: $O(n^2)$ word complexity

2. Committee sampling
   • Cryptographic sortition
   • Reduces word complexity to $O(n \log n)$

Following Algorand [Gilad et al. 2017]
Committee Sampling

• Use the VRF to sample $O(\log n)$ processes to a committee in each round
• Replace all-to-all rounds with committee-to-all rounds

• Evading the adversary:
  • Use a new committee in each round
  • Send to all since committees are unpredictable
  • By Chernoff bounds, “not too many” faulty processes in each committee
Shared Coin – Take 2

\[ v_1 \xrightarrow{\text{first, } v_i} \] \[ \text{return LSB of minimum value} \]
Word complexity: $O(n \log n)$
Word complexity: $O(n \log n)$
But how many messages do we wait for?
Committee Sampling in Asynchronous Model

• Committee based protocols cannot wait for $n - f$ processes. Instead, they wait for $W$ processes.

• We choose $W, B$ so that using Chernoff bounds, WHP:
  1. At least $W$ processes in each committee are correct
  2. At most $B$ processes in each committee are Byzantine
Committee Sampling in Asynchronous Model

3. Every two subsets in a committee of size $W$ intersect by at least $B + 1$ processes

4. Every two subsets in a committee of size $W$ and $B + 1$ intersect by at least 1 process
Shir Cohen’s Shared Coin

wait for $W$ messages

wait for $W$ messages

return LSB of minimum value
From Coin Flipping to (Binary) BA WHP

- Approver based on [Bracha 1987] – reliable broadcast
  - But with committee sampling
- BA based on [Mostefaoui et al. 2015]
API: \textit{approve}_i(v_i) returns a set of values

We assume \textit{approve} is called with at most two different values

WHP the following hold:

• \textbf{Validity}: If all correct processes invoke \textit{approve}(v) then the only possible return value of correct processes is \{v\}

• \textbf{Graded agreement}: If correct processes return both \{v\} and \{w\} then \(v = w\)

• \textbf{Termination}: If all correct processes invoke \textit{approve} then it returns with a non-empty set at all of them
Approver 🌟 Without Sampling

Echo $v$ upon receiving $f+1$ $v$

Send $\langle ok, v \rangle$ with $n-f$ signatures upon receiving $n-f$ $\langle echo, v \rangle$

Return the set of values in the first $n-f$ ok messages

May speak twice
Approver 👍 With Sampling

Echo $v$ upon receiving $B+1$ $v$

Send $<\text{ok}, v>$ with $W$ signatures upon receiving $W <\text{echo}, v>$

Return the set of values in the first $W$ ok messages
Approver 👍 With Sampling

Word complexity: $O(n \log^2 n)$

Send <ok, v> with W signatures upon receiving W <echo, v>

Return the set of values in the first W ok messages
From Coin Flipping to (Binary) BA WHP

- Approver based on [Bracha 1987] – reliable broadcast
  - But with committee sampling
- BA based on [Mostefaoui et al. 2015]
$\textbf{BA WHP}$

1: $est_i \leftarrow v_i$
2: $decision_i \leftarrow \bot$

3: for $r = 0, 1, \ldots$ do
4: $vals \leftarrow \text{approve}(est_i)$
5: if $vals = \{v\}$ for some $v$ then
6: $propose_i \leftarrow v$
7: otherwise $propose_i \leftarrow \bot$
8: $c \leftarrow \text{whp\_coin}(r)$

9: $props \leftarrow \text{approve}(propose_i)$
10: if $props = \{v\}$ for some $v \neq \bot$ then
11: $est_i \leftarrow v$
12: if $decision_i = \bot$ then
13: $decision_i \leftarrow v$
14: else
15: if $props = \{\bot\}$ then
16: $est_i \leftarrow c$
17: else $\%props = \{v, \bot\}$
18: $est_i \leftarrow v$
BA WHP

1: \( est_i \leftarrow v_i \)
2: \( decision_i \leftarrow \bot \)
3: \textbf{for} \( r = 0, 1, \ldots \) \textbf{do}
4: \( \text{vals} \leftarrow \text{approve}(est_i) \)
5: \textbf{if} \( \text{vals} = \{v\} \) \textbf{for some} \( v \) \textbf{then}
6: \( \text{propose}_i \leftarrow v \)
7: \textbf{otherwise} \( \text{propose}_i \leftarrow \bot \)
8: \( c \leftarrow \text{whp\_coin}(r) \)
9: \( \text{props} \leftarrow \text{approve}(\text{propose}_i) \)
10: \textbf{if} \( \text{props} = \{v\} \) \textbf{for some} \( v \neq \bot \) \textbf{then}
11: \( est_i \leftarrow v \)
12: \textbf{if} \( \text{decision}_i = \bot \) \textbf{then}
13: \( \text{decision}_i \leftarrow v \)
14: \textbf{else}
15: \textbf{if} \( \text{props} = \{\bot\} \) \textbf{then}
16: \( est_i \leftarrow c \)
17: \textbf{else} \( \%\text{props} = \{v, \bot\} \)
18: \( est_i \leftarrow v \)
BA WHP

1: \( est_i \leftarrow v_i \)
2: \( \text{decision}_i \leftarrow \bot \)

3: \( \text{if } vals = \{v\} \text{ for some } v \neq \bot \) then
4: \( \quad \text{propose}_i \leftarrow v \)
5: \( \quad \text{otherwise propose}_i \leftarrow \bot \)
6: \( c \leftarrow \text{whp\_coin}(\tau) \)
7: \( \quad \text{if props} = \{\bot\} \) then
8: \( \quad \quad \text{est}_i \leftarrow c \)
9: \( \quad \quad \text{if props} = \{v, \bot\} \) else
10: \( \quad \quad \quad \text{est}_i \leftarrow v \)
11: \( \quad \text{else} %props = \{v, \bot\} \)
12: \( \quad \quad \text{est}_i \leftarrow v \)
13: \( \text{end if} \)
14: \( \text{end if} \)

Word complexity: \( O(n \log^2 n) \)
Not a COINcidence Summary

• First formalization of randomly sampled committees using cryptography in asynchronous settings
• First sub-quadratic asynchronous shared coin and BA WHP algorithms
• Expected $\tilde{O}(n)$ word complexity and $O(1)$ expected time

Limitations:
• Binary consensus only
• Safety and liveness only WHP
• One-shot algorithm (not BAB/SMR)
• Non-optimal resilience – improved by [Blum et al. 2020]
Making It Scale

- Solve BA with high probability (WHP) (probability of being correct tends to 1 as $n \to \infty$)
- Complexity: $\tilde{O}(n)$

VRFs

Perfect coin

- Solve BAB with deterministic safety, probabilistic liveness
- *Amortized* complexity: $O(n)$ or $O(n \log n)$ per agreement
All You Need is DAG

Idit Keidar
Technion

Oded Naor
Technion

Eleftherios Kokoris-Kogias
Novi Research

Alexander Spiegelman
Novi Research

PODC 2021
Model

• Asynchronous
• Byzantine faults, optimal resilience: $f < \frac{n}{3}$

• Crypto: PKI, threshold signatures
  • Can implement global perfect coin using secret sharing
  • Always safe (information theoretically)
  • Unpredictable to a computationally bounded adversary
BAB: Byzantine Atomic Broadcast

propose a transaction

babcast

deliver

deliver

deliver

deliver

BAB

apply to the state machine
BAB: Byzantine Atomic Broadcast

propose a transaction

.deliver
.bcast

... deliver

apply to the ledger
BAB: Byzantine Atomic Broadcast

- All messages sent by correct processes are eventually delivered (fairness)
- Correct processes deliver the same messages
  - possibly in different orders
- Liveness with probability 1

= Reliable Broadcast + Total Order

- Correct processes deliver messages in the same order
Building Blocks

• Reliable broadcast
  • Bracha broadcast
  • Guerraoui et al. (gossip-based, success probability $1-\varepsilon$)
  • Cachin & Tessaro verifiable dispersal

• Unpredictable leader election
  • Based on the perfect coin
  • Same sequence of leaders at all nodes – information theoretically secure
  • Unpredictable by a bounded adversary – needed only for liveness
DAG-Rider

1. Build a DAG
   • Representing causal dependencies among messages
   • Reliably broadcast the DAG
   • ~30 lines of pseudo-code

2. Elect random leaders (vertices in the DAG)
   • Use the DAG to generate shared randomness

3. Order Messages
   • Sequence the leaders’ entire causal history
   • Locally, zero overhead
   • ~30 lines of pseudo-code
DAG-Rider

1. Build a DAG
   • Representing causal dependencies among messages
   • Reliably broadcast the DAG
   • ~30 lines of pseudo-code

2. Elect random leaders (vertices in the DAG)
   • Use the DAG to generate shared randomness

3. Order Messages
   • Sequence the leaders’ entire causal history
   • Locally, zero overhead
   • ~30 lines of pseudo-code
1. Build a DAG

On deliver n-f messages, bcast next message

Round 0  Round 1

$p_1$’s messages:

$p_2$’s messages:

$p_3$’s messages:

$p_4$’s messages:
The DAG

1. Source & round
2. A set of values
3. n-f strong edges
4. Weak edges (for fairness)
Reliably Broadcast the DAG

• No equivocation: all processes see the same DAG, eventually
• But their partial views may temporarily differ

• Complexity:
  • Each vertex includes a linear number of edges
  • Propose $\Omega(n)$ messages in each vertex to amortize costs
  • Reliable broadcast cost depends on underlying protocol
DAG-Rider

1. Build a DAG
   • Representing causal dependencies among messages
   • Reliably broadcast the DAG
   • ~30 lines of pseudo-code

2. Elect random leaders (vertices in the DAG)
   • Use the DAG to generate shared randomness

3. Order Messages
   • Sequence the leaders’ entire causal history
   • Locally, zero overhead
   • ~30 lines of pseudo-code
2. Elect Leaders

• Divide DAG into *waves* of four rounds
• Use round i+3 messages to elect a leader in round i

![Diagram](image)

threshold signature for unpredictable leader election
DAG-Rider

1. Build a DAG
   • Representing causal dependencies among messages
   • Reliably broadcast the DAG
   • ~30 lines of pseudo-code

2. Elect random leaders (vertices in the DAG)
   • Use the DAG to generate shared randomness

3. Order Messages
   • Sequence the leaders’ entire causal history
   • Locally, zero overhead
   • ~30 lines of pseudo-code
3. Order Messages

• Based on the local view of the DAG + elected leaders
• Because processes may have different views, *commit* only leaders that have sufficient support
  • Appear in sufficiently many DAGs
Divide the DAG into Waves
Wave Leader Commit Rule

• **Commit** the wave’s elected leader if there are $2f+1$ vertices in the 4th round with strong paths to it
  • Weak edges do not count

Every vertex in the next wave will have a strong path to the leader
Leveraging the Common Core Principle

“After three rounds of all-to-all sending and collecting accumulated sets of values from \( n-f \) processes, all correct processes have at least \( 2f+1 \) common values”

\[ \therefore \text{At least } 2f+1 \text{ round 1 vertices satisfy the commit rule} \]

\[ \therefore \text{The elected vertex satisfies it with probability at least } 2/3 \]
  - It is unpredictable until the end of round 4

\[ \therefore \text{O(1) expected latency until a leader is committed} \]
What About Uncommitted Leaders?

• Another process may have committed wave i
Committing the Same Leaders

• When committing a leader in wave i, check if any previous leaders need to be committed first

The commit rule guarantees a path between every pair of committed leaders
Sequencing Messages

• For each leader, sequence its entire causal history
• Weak edges count
# DAG-Rider Variants vs. Previous Work

<table>
<thead>
<tr>
<th>Variant Description</th>
<th>Communication</th>
<th>Latency</th>
<th>Fairness</th>
<th>Post-quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>VABA SMR</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>no</td>
<td>Unsafe</td>
</tr>
<tr>
<td>Dumbo SMR</td>
<td>amortized $O(n)$</td>
<td>$O(\log n)$</td>
<td>no</td>
<td>Unsafe</td>
</tr>
<tr>
<td>DAG-Rider + Bracha</td>
<td>amortized $O(n^2)$</td>
<td>$O(1)$</td>
<td>with probability 1</td>
<td>Safe</td>
</tr>
<tr>
<td>DAG-Rider + Gossip</td>
<td>amortized $O(n \log n)$</td>
<td>$O\left(\frac{\log n}{\log \log n}\right)$</td>
<td>with probability $1-\varepsilon$</td>
<td>Safe</td>
</tr>
<tr>
<td>DAG-Rider + verifiable dispersal</td>
<td>amortized $O(n)$</td>
<td>$O(1)$</td>
<td>with probability 1</td>
<td>Safe</td>
</tr>
</tbody>
</table>
DAG-Rider Summary

• O(1) expected latency
• O(n) amortized message complexity per agreement
  • With appropriate reliable broadcast and batching
• Post-quantum safe
  • Use crypto only for leader unpredictability
• Fair with probability 1
  • Every message sent by a correct process is eventually delivered