Order-Fairness for Byzantine Consensus

Mahima Kelkar, Fan Zhang, Steven Goldfeder, Ari Juels

June 8, 2023

Introduction

In any consensus (or State Machine Replication) protocol, the following two properties must be satisfied:

- **Consistency**:all honest nodes must have the same view of the agreed upon log
- **Liveness**: messages submitted by users are added to the log within a reasonable amount of time

Introduction

In any consensus (or State Machine Replication) protocol, the following two properties must be satisfied:

- **Consistency**:all honest nodes must have the same view of the agreed upon log
- **Liveness**: messages submitted by users are added to the log within a reasonable amount of time

The novel idea that Kelkar et al formulated in this paper is a property called **transaction order fairness**: if a (sufficiently) large number of nodes receive a transaction tx_1 before another one tx_2, then this should be reflected in the final ordering agreed upon by all nodes.

Importance of fair transaction ordering

To deter attacks based on the manipulation of the order in which transactions are included in each block. Namely as **front running** and **sandwich attacks**. For example in the Ethereum network, such practices are commonplace and have cost its users millions of dollars.

Our Model

We assume the following for our Model:

- **Permissioned setting**: the number of consensus nodes n (validators) and their identities is known and fixed. The users can be arbitrarily many.
- **Byzantine Adversary:** a type of active adversary that can deviate from the prescribed protocol, send conflicting messages to different nodes, or delay any message from a node they control.

Our Model

We assume the following for our Model:

- **Permissioned setting**: the number of consensus nodes n (validators) and their identities is known and fixed. The users can be arbitrarily many.
- **Byzantine Adversary:** a type of active adversary that can deviate from the prescribed protocol, send conflicting messages to different nodes, or delay any message from a node they control.
- **Synchronization:** Kelkar et al designed protocols in both synchronous and asynchronous settings.
  - Synchronous: there is a well-defined and known upper bound on message delivery time.
  - Asynchronous: there are no assumptions made about timing bounds.

Defining Fair Ordering

How do we go about defining the desired property?

Definition

Send-order-fairness: if tx_1 was sent by a user before tx_2, then it will have to appear this way in the log.

Defining Fair Ordering

How do we go about defining the desired property?

Definition

Send-order-fairness: if tx_1 was sent by a user before tx_2, then it will have to appear this way in the log.

To achieve this, we need the following two strong assumptions:

1. A trusted way to timestamp a transaction at the user side, so a node can not delay the relay of a transaction.
2. Network synchrony is also required to ensure that a transaction is not arbitrarily delayed (by offline nodes or by the Adversary).

The above assumptions are quite strong, so a different approach has to be made.

Defining Fair Ordering

Definition

Receive-order-fairness: If sufficiently many (at least γ-fraction) nodes receive a transaction tx_1 before another transaction tx_2, then all honest nodes must output tx_1 before tx_2, where $\frac{1}{2} < \gamma \leq 1$ is the order fairness parameter of the protocol.

Defining Fair Ordering

Definition

Receive-order-fairness: If sufficiently many (at least γ-fraction) nodes receive a transaction tx_1 before another transaction tx_2, then all honest nodes must output tx_1 before tx_2, where $\frac{1}{2} < \gamma \leq 1$ is the order fairness parameter of the protocol.

Impossibility result: Receive-order-fairness is impossible to be achieved, unless we assume:

- $\gamma = 1$.
- Non-corrupting (passive) Adversary.
- External synchrony assumption (i.e.instant synchronous external network).

Condorcet Paradox

Consider the following case for 3 nodes A, B, C:

- Node A receives $tx_1 < tx_2 < tx_3$
- Node B receives $tx_2 < tx_3 < tx_1$
- Node C receives $tx_3 < tx_1 < tx_2$

Condorcet Paradox

Consider the following case for 3 nodes A, B, C:

- Node A receives $tx_1 < tx_2 < tx_3$
- Node B receives $tx_2 < tx_3 < tx_1$
- Node C receives $tx_3 < tx_1 < tx_2$

No protocol can satisfy this property for $\gamma \leq \frac{2}{3}$, since such a protocol would have to include tx_1 before tx_2, tx_2 before tx_3, and tx_3 before tx_1 in its final log.

Condorcet Paradox

Consider the following case for 3 nodes A, B, C:

- Node A receives $tx_1 < tx_2 < tx_3$
- Node B receives $tx_2 < tx_3 < tx_1$
- Node C receives $tx_3 < tx_1 < tx_2$

No protocol can satisfy this property for $\gamma \leq \frac{2}{3}$, since such a protocol would have to include tx_1 before tx_2, tx_2 before tx_3, and tx_3 before tx_1 in its final log.
We can extent the above for $\gamma \leq \frac{n-1}{n}$, without any Adversarial presence.

Condorcet Paradox

Consider the following case for 3 nodes A, B, C:

- Node A receives $tx_1 < tx_2 < tx_3$
- Node B receives $tx_2 < tx_3 < tx_1$
- Node C receives $tx_3 < tx_1 < tx_2$

No protocol can satisfy this property for $\gamma \leq \frac{2}{3}$, since such a protocol would have to include tx_1 before tx_2, tx_2 before tx_3, and tx_3 before tx_1 in its final log.
We can extent the above for $\gamma \leq \frac{n-1}{n}$, without any Adversarial presence.
For $\gamma = 1$, the Adversary has to corrupt a single node in order to prevent this property.

Block-order-fairness

Definition

Block-order-fairness: If sufficiently many (at least γ-fraction) nodes receive a transaction tx_1 before another transaction tx_2, then no honest node can deliver tx_1 in a block after tx_2.

This relaxation allows us to evade the Condorcet paradox by a simple trick: placing paradoxical orderings into the same block.

Block-order-fairness

Definition

Block-order-fairness: If sufficiently many (at least γ-fraction) nodes receive a transaction tx_1 before another transaction tx_2, then no honest node can deliver tx_1 in a block after tx_2.

This relaxation allows us to evade the Condorcet paradox by a simple trick: placing paradoxical orderings into the same block. The Aequitas protocol designed by Kelkar et al achieves this property. The remaining part of the presentation will be dedicated to describing the Aequitas protocol in the leader-based, synchronous setting.

Aequitas: an overview

The Aequitas protocols follow a three-step process. Every transaction *tx*, undergoes three stages:

1. **Gossip Stage**: Nodes gossip transactions in the order it received them. Thus, each node gossips its *local transaction ordering*.
2. **Agreement Stage**: Nodes agree on the set of nodes whose local orderings should be considered for deciding on the *global ordering* of a particular transaction.
3. **Finalization Stage**: Nodes finalize the global ordering of a transaction using the set of *local orderings* decided on in the agreement stage.

FIFO Broadcast

Definition

A FIFO-BC protocol satisfies liveness, agreement and FIFO-order if the following properties hold with overwhelming probability:

1. $(T_{warmup}, T_{confirm})$-Liveness: If the sender is honest and receives m in round $r > T_{warmup}$, or if an honest node delivers m in round $r > T_{warmup}$ then all honest nodes will have delivered m by round $r + T_{confirm}$. .
2. Agreement: If an honest node delivers m before m', then no honest node delivers m' unless it has already delivered m.
3. FIFO-Order: If the sender is honest and is input a message m before m', then no honest node delivers m' unless it has already delivered m.

FIFO Broadcast

Remarks:

- When composing several FIFO broadcast primitives together with different senders, FIFO order is maintained for each individual sender but different honest nodes may deliver messages from different senders in different orders.
- FIFO broadcast can be achieved using reliable broadcast: sequence numbers are added to the messages broadcast by the sender in a reliable broadcast protocol. An honest node does not deliver a message with sequence number k until it has delivered a message with sequence number k-1.

Gossip Stage

Based FIFO-BC. FIFO-BC guarantees that broadcasts by an honest node are delivered by other honest nodes in the same order that they were received.
Let Log_i^j be node i's view of the order in which node j received transactions, according to how node j gossiped them.If node j is malicious, Log_i^j may arbitrarily differ from the actual order in which j received transactions, but FIFO-BC prevents j from equivocating, i.e., any two honest nodes i and k will have consistent Log_i^j and Log_k^j.

Aequitas

1. **Gossip Stage**: Nodes gossip transactions in the order it received them. Thus, each node gossips its *local transaction ordering*. Achieved by using **FIFO Broadcast**.
2. **Agreement Stage**: Nodes agree on the set of nodes whose local orderings should be considered for deciding on the *global ordering* of a particular transaction.
3. **Finalization Stage**: Nodes finalize the global ordering of a transaction using the set of *local orderings* decided on in the agreement stage.

Set Byzantine Agreement

Definition

A set-BA protocol satisfies agreement, inclusion validity and exclusion validity, if the following properties hold with overwhelming probability:

1. Agreement: If two honest nodes i and j output set O_i and $O_j \Rightarrow O_i = O_j$
2. Inclusion Validity: If an element is in the input sets of all nodes, then it will also be in the output sets of all honest nodes. $\forall i \in P : c \in U_i \Rightarrow c \in O_j$ for all honest j.
3. If an element is not in any input set, then it is not in any honest output set. $\forall i \in P : c \notin U_i \Rightarrow c \notin O_j$ for all honest j.

Set Byzantine Agreement

Remarks:

- Set-Ba can be realized by BBA (Binary Byzantine Agreement) protocol:
  - $\forall i \in P \forall c \in U_i : \prod_{BBA}(c)$
  - Collect the outputs for all accepting instances in the set $O = \{c | \prod_{BBA}(c) = 1\}$.
- Set-BA inherits validity and liveness for BBA.

Agreement Stage

At the end of the gossip stage for a transaction tx, a node i ends up with a set of other nodes U_i^{tx}, that included tx in their local ordering. Hence, $tx \in Log_i^k \Rightarrow k \in U_i^{tx}$
Nodes now perform Set-BA to agree on a set L^{tx}, to agree on the final set of transactions that every honest node has heard.

Aequitas

1. **Gossip Stage**: Nodes gossip transactions in the order it received them. Thus, each node gossips its *local transaction ordering*. Achieved by using **FIFO Broadcast**.
2. **Agreement Stage**: Nodes agree on the set of nodes whose local orderings should be considered for deciding on the *global ordering* of a particular transaction. Achieved by using **Set-BA**.
3. **Finalization Stage**: Nodes finalize the global ordering of a transaction using the set of *local orderings* decided on in the agreement stage.

Waiting Graph

- A node i maintains a directed graph G_i, where vertices represent transactions and an edge from a to b denotes that b is waiting for a to be delivered. G_i is the "waiting graph" maintained by node i.

Waiting Graph

- A node i maintains a directed graph G_i, where vertices represent transactions and an edge from a to b denotes that b is waiting for a to be delivered. G_i is the "waiting graph" maintained by node i.
- Since nodes are building this graph on the same "data" (the set of local logs agreed upon in the agreement phase), nodes will have consistent graphs. That is, if an edge (a, b) exists in G_i, then it will also (eventually) exist in G_j, if i and j are both honest.

Waiting Graph

- A node i maintains a directed graph G_i, where vertices represent transactions and an edge from a to b denotes that b is waiting for a to be delivered. G_i is the "waiting graph" maintained by node i.
- Since nodes are building this graph on the same "data" (the set of local logs agreed upon in the agreement phase), nodes will have consistent graphs. That is, if an edge (a, b) exists in G_i, then it will also (eventually) exist in G_j, if i and j are both honest.
- "Paradoxical" transactions will cause circles in G_i.

Waiting Graph

- A node i maintains a directed graph G_i, where vertices represent transactions and an edge from a to b denotes that b is waiting for a to be delivered. G_i is the "waiting graph" maintained by node i.
- Since nodes are building this graph on the same "data" (the set of local logs agreed upon in the agreement phase), nodes will have consistent graphs. That is, if an edge (a, b) exists in G_i, then it will also (eventually) exist in G_j, if i and j are both honest.
- "Paradoxical" transactions will cause circles in G_i.
- The condensation graph, collapses the strongly connected components in G_i into the same vertex. Each vertex now represents a set of transactions, that will be in the same block. Since the condensation graph is acyclic, a total ordering can be extracted.

Finalization Stage

Building the waiting graph:
Let $l_{(tx,tx')}$ be the number of of logs where tx was ordered before tx'. If this is small. then tx should wait until tx' is finalized.

Finalization Stage

Building the waiting graph:
Let $l_{(tx,tx')}$ be the number of of logs where tx was ordered before tx'. If this is small. then tx should wait until tx' is finalized. Hence, an edge tx, tx' is added in G_i if

$$l_{tx,tx'} \leq \left|L^{tx} \cup L^{tx'}\right| - \gamma \cdot n + f$$

Finalization Stage

Building the waiting graph:
Let $l_{(tx,tx')}$ be the number of of logs where tx was ordered before tx'. If this is small. then tx should wait until tx' is finalized. Hence, an edge tx, tx' is added in G_i if

$$l_{tx,tx'} \leq \left|L^{tx} \cup L^{tx'}\right| - \gamma \cdot n + f$$

Finally the leader proposes a set of transactions. Each node i validates the proposal if each transaction in the set belongs in the same strongly connected component in each G_i.

Aequitas

1. **Gossip Stage**: Nodes gossip transactions in the order it received them. Thus, each node gossips its *local transaction ordering*. Achieved by using **FIFO Broadcast**.
2. **Agreement Stage**: Nodes agree on the set of nodes whose local orderings should be considered for deciding on the *global ordering* of a particular transaction. Achieved by using **Set-BA**.
3. **Finalization Stage**: Nodes finalize the global ordering of a transaction using the set of *local orderings* decided on in the agreement stage. Achieved by using the **Waiting Graph**.

Overall Results

The Aequitas protocol only rely on reliable broadcast and Byzantine agreement, both of which can be realized by any existing consensus protocol. Hence, any existing protocol can be extended achieve block order fairness.

Protocol	Style	Network	Corruption Bound†	Consistency	Liveness	Order-Fairness
$\Pi^{\text{sync,lead}}_{\text{Aequitas}}$	Leader	Synchronous*	$n > \frac{2f}{2\gamma-1}$	✓	✓ (Weak)	✓
$\Pi^{\text{sync,nolead}}_{\text{Aequitas}}$	Leaderless	Synchronous*	$n > \frac{2f}{2\gamma-1}$	✓	✓ (Weak)	✓
$\Pi^{\text{async,lead}}_{\text{Aequitas}}$	Leader	Any	$n > \frac{4f}{2\gamma-1}$	✓	✓ (Eventual, Weak)	✓
$\Pi^{\text{async,nolead}}_{\text{Aequitas}}$	Leaderless	Any	$n > \frac{4f}{2\gamma-1}$	✓	✓ (Eventual, Weak)	✓

* Completely Synchronous Setting (See Section 2)

† $\frac{1}{2} < \gamma \leq 1$ is the order-fairness parameter (See Section 4)