TUTORIAL:
An In-Depth Look at BFT Consensus in Blockchains: Challenges and Opportunities (Theory)

Suyash Gupta  
Sajjad Rahnama

Jelle Hellings  
Mohammad Sadoghi

Exploratory Systems Lab,  
Department of Computer Science,  
University of California, Davis, CA, USA
Introduction to Blockchains: Theory on resilient fully-replicated systems
What is a Blockchain?

A resilient tamper-proof append-only sequence of transactions maintained by many participants.
What is a Blockchain?

A **resilient** tamper-proof append-only sequence of transactions maintained by many participants.

▶ **Resilient.**
   High availability via full replication among participants.
What is a Blockchain?

A resilient tamper-proof append-only sequence of transactions maintained by many participants.

- **Resilient.**
  High availability via full replication among participants.

- **Tamper-proof.**
  Changes can only be made with majority participation.
What is a Blockchain?

A resilient tamper-proof **append-only sequence of transactions** maintained by many participants.

- **Resilient.**
  High availability via full replication among participants.

- **Tamper-proof.**
  Changes can only be made with majority participation.

- **Append-only sequence of transactions.**
  In database terms: a journal or log.
What is a Blockchain?

A resilient tamper-proof append-only sequence of transactions maintained by many participants.

- **Resilient.**
  High availability via full replication among participants.

- **Tamper-proof.**
  Changes can only be made with majority participation.

- **Append-only sequence of transactions.**
  In database terms: a journal or log.

Basic Blockchains are *distributed fully-replicated systems!*
Blockchain technology: Many terms

1. Permissionless versus permissioned.
2. Distributed fully-replicated systems: CAP Theorem.
3. Crash tolerance versus Byzantine fault tolerance.
4. Consensus, broadcast, interactive consistency.
5. Synchronous versus asynchronous communication.

Main focus today

Permissioned, Byzantine Fault tolerance, Asynchronous.
### Membership: Permissionless versus permissioned

<table>
<thead>
<tr>
<th>Permissionless</th>
<th>Permissioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participants are not known.</td>
<td>Traditional resilient systems (PBFT, …)</td>
</tr>
<tr>
<td>Can provide <em>open membership.</em></td>
<td>ResilientDB</td>
</tr>
<tr>
<td>Permissioned Participants are known and vetted.</td>
<td>HyperLedger</td>
</tr>
</tbody>
</table>

- Public Blockchains
- Bitcoin
- Ethereum
- …
Membership: Tamper-proof structures

How is the Blockchain made tamper-proof?

**Permissionless**  
Additions and changes cost *resources*.
Tamper-proof: the majority of resources behave!

\[
\begin{array}{c}
| h_0 \ p_1 \ | \\
| T_1 |
\end{array} 
\begin{array}{c}
| h_1 \ p_2 \ | \\
| T_2 |
\end{array} 
\begin{array}{c}
| h_2 \ p_3 \ | \\
| T_3 |
\end{array}
\]

**Permissioned**  
Additions and changes are *authenticated*.
Tamper-proof: the majority of participants behave!

\[
\begin{array}{c}
| S_{11}, \ldots, S_{1p} \ | \\
| T_1 |
\end{array} 
\begin{array}{c}
| S_{21}, \ldots, S_{2p} \ | \\
| T_2 |
\end{array} 
\begin{array}{c}
| S_{31}, \ldots, S_{3p} \ | \\
| T_3 |
\end{array}
\]

In both cases: reliance on strong cryptography!
Distributed fully-replicated systems

**Consistency**  Does every participant have exactly the same data?
**Availability**  Does the system continuously provide services?
**Partitioning**  Can the system cope with network disturbances?

**Theorem (The CAP Theorem)**

*Can provide at most two-out-of-three of these properties.*
Distributed fully-replicated systems

**Consistency**  Does every participant have exactly the same data?
**Availability**  Does the system continuously provide services?
**Partitioning**  Can the system cope with network disturbances?

**Theorem (The CAP Theorem)**

*Can provide at most two-out-of-three of these properties.*

**CAP Theorem uses narrow definitions!**
The CAP Theorem and Blockchains

Consistency

Availability

Partitioning
Permissionless Blockchains
Open membership focuses on Availability and Partitioning.
⇒ Consistency not guaranteed (e.g., forks).
The CAP Theorem and Blockchains

Permissioned Blockchains
Consistency at all costs.

⇒ Availability when communication is reliable.
Consistency: 2PC, 3PC, Paxos, Consensus
Consensus in permissioned Blockchains

A *consensus algorithm* is an algorithm satisfying:

Termination  Each non-faulty replica decides on a transaction. CAP: availability, a *liveness* property.

Non-divergence Non-faulty replicas decide on the same transaction. CAP: consistency, a *safety* property.
Consensus in permissioned Blockchains

A *consensus algorithm* is an algorithm satisfying:

**Termination** Each non-faulty replica decides on a transaction.
- CAP: availability, a *liveness* property.

**Non-divergence** Non-faulty replicas decide on the same transaction.
- CAP: consistency, a *safety* property.

Blockchains provide *client-server services*:

**Validity** Every decided-on transaction is a client request.
**Response** Clients learn about the outcome of their requests.
**Service** Every client will be able to request transactions.
Consensus in permissioned Blockchains

A *consensus algorithm* is an algorithm satisfying:

**Termination** Each non-faulty replica decides on a transaction.
CAP: availability, a *liveness* property.

**Non-divergence** Non-faulty replicas decide on the same transaction.
CAP: consistency, a *safety* property.

Blockchains provide *client-server services*:

**Validity** Every decided-on transaction is a client request.

**Response** Clients learn about the outcome of their requests.

**Service** Every client will be able to request transactions.

![Diagram of consensus in permissioned Blockchains]

- **c**: Client
- **R1, R2, R3, R4**: Replicas
- **Consensus**: Agreement process
- **T**: Transaction
- **O**: Output

UC Davis

10/52
From consensus to a consistent Blockchain

Reminder: append-only sequence of transactions.

1. Decide on transactions in rounds.
2. In round $\rho$, use consensus to select a client transaction $T$.
3. Append $T$ as the $\rho$-th entry to the Blockchain.
4. Execute $T$ as the $\rho$-th entry, inform client.

Consistent state: linearizable order and deterministic execution
On identical inputs, execution of transactions at all non-faulty replicas must produce identical outputs.
Byzantine Broadcast (Generals)

Assume a replica $G$ is the general and holds transaction $T$.

A **Byzantine broadcast algorithm** is an algorithm satisfying:

- **Termination** Each non-faulty replica decides on a transaction.
- **Non-divergence** Non-faulty replicas decide on the same transaction.
- **Dependence** If the general $G$ is non-faulty, then non-faulty replicas will decide on $T$.

$\overset{T}{\longrightarrow} \overset{T'}{\longrightarrow} \overset{T'}{\longrightarrow} \overset{T'}{\longrightarrow}$

($T' = T$ if the general $G$ is non-faulty).
Interactive consistency

Assume \( n \) replicas and each replica \( R_i \) holds a transaction \( T_i \). An *interactive consistency algorithm* is an algorithm satisfying:

**Termination** Each non-faulty replica decides on \( n \) transactions.

**Non-divergence** Non-faulty replicas decide on the same transactions.

**Dependence** If replica \( R_j \) is non-faulty, then non-faulty replicas will decide on \( T_j \).

![Diagram showing interactive consistency with replicas and transactions](image-url)

(As \( R_3 \) is faulty: \( \times \) can be anything)
Theory of Byzantine systems

Many theoretical results!

1. Failure model: crashes and Byzantine failures.
2. Synchronous versus asynchronous communication.
3. Digital signatures versus authenticated communication.
4. Lower bounds on communication (phases, messages).
5. Connectivity of the replicas and quality of the network.
Failure model: Crashes and Byzantine failures

Crash  Participant stops participating in the system.
Byzantine  Participant behaves arbitrary.
          Participants can be *coordinated malicious*.

We need assumptions!
If all participants crash or are malicious, no service can be provided.

<table>
<thead>
<tr>
<th>Permissionless</th>
<th>Permissioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic primitives</td>
<td>Cryptographic primitives</td>
</tr>
<tr>
<td>Majority of resources</td>
<td>Majority of participants</td>
</tr>
</tbody>
</table>
Synchronous versus asynchronous communication

**Synchronous**  Reliable communication with bounded delays.

**Asynchronous**  Unreliable communication:
message loss, arbitrary delays, duplications, …

Theorem (Fisher, Lynch, and Paterson)
*There exists no asynchronous 1-crash-resilient consensus algorithm.*
Synchronous versus asynchronous communication

Synchronous  Reliable communication with bounded delays.
Asynchronous  Unreliable communication:
               message loss, arbitrary delays, duplications, …

Theorem (Fisher, Lynch, and Paterson)

There exists no asynchronous 1-crash-resilient consensus algorithm.

Asynchronous consensus
Assuming synchronous communication is often not practical.

Termination  Reliable communication/probabilistic.
Non-divergence  Always guaranteed.
Digital signatures versus authenticated communication

- Digital signatures via *public-key cryptography*. Byzantine replicas cannot tamper with forwarded messages.
- Authenticated communication via *message authentication codes*. Byzantine replicas are only able to impersonate each other. Cannot impersonate non-faulty replicas.

Theorem (Pease, Shostak, and Lamport)

*Assume a system with n replicas of which at most f are Byzantine.*

1. *In general, broadcast protocols require n > 3f.*
2. *Synchronous communication and digital signatures: n > f.*
Digital signatures versus authenticated communication

- Digital signatures via *public-key cryptography*. Byzantine replicas cannot tamper with forwarded messages.
- Authenticated communication via *message authentication codes*. Byzantine replicas are only able to impersonate each other. Cannot impersonate non-faulty replicas.

Theorem (Pease, Shostak, and Lamport)

Assume a system with $n$ replicas of which at most $f$ are Byzantine.

1. In general, broadcast protocols require $n > 3f$.
2. Synchronous communication and digital signatures: $n > f$.

Bounds for consensus: response via majority votes
For clients to learn outcome, we require at least $n > 2f$. 
Theorem (Fisher and Lynch; Dolev, Reischuk, and Strong)

Assume a system with \( n \) replicas of which at most \( f \) can be Byzantine.

1. **Consensus**: worst-case \( \Omega(f + 1) \) phases of communication.
2. **Optimistic Broadcasts**: \( \Omega(t + 2) \) phases if \( t \leq f \) failures happen.
Lower bounds on communication (phases, messages)

Theorem (Fisher and Lynch; Dolev, Reischuk, and Strong)
Assume a system with $n$ replicas of which at most $f$ can be Byzantine.
1. Consensus: worst-case $\Omega ( f + 1 )$ phases of communication.
2. Optimistic Broadcasts: $\Omega ( t + 2 )$ phases if $t \leq f$ failures happen.

Theorem (Dolev and Reischuk)
Assume a system with $n$ replicas of which at most $f$ can be Byzantine.
Any broadcast protocol using digital signatures requires:
1. $\Omega ( nf )$ digital signatures;
2. $\Omega ( n + f^2 )$ messages.
Connectivity of the replicas and quality of the network

Theorem (Dolev)

Assume a system with \( n \) replicas of which at most \( f \) can be Byzantine. Broadcast: the network must stay connected when removing \( 2f \) replicas.
Connectivity of the replicas and quality of the network

Theorem (Dolev)

Assume a system with $n$ replicas of which at most $f$ can be Byzantine. Broadcast: the network must stay connected when removing $2f$ replicas.

Network assumptions in practice

- Clique: direct communication between all replica pairs.
- Gossip: needs some network quality.
Theory of Byzantine systems: Summary

Limitations of practical consensus algorithm:

- Dealing with $f$ malicious failures requires $n > 3f$ replicas.
- Worst-case: at least $\Omega (f + 1)$ phases of communication.
- Worst-case: at least $\Omega (nf)$ signatures and $\Omega (n + f^2)$ messages.
- Termination: reliable communication
  - Between most replicas;
  - Communication with bounded-delay.
A practical consensus protocol: PBFT
**PBFT: Practical Byzantine Fault Tolerance**

**Primary** Coordinates consensus: propose transactions to replicate.

**Backup** Accept transactions and verifies behavior of primary.
PBFT: Normal-case protocol in view $v$

\[ \langle T \rangle_c. \]
**PBFT**: Normal-case protocol in view \( v \)

\[
\text{PrePrepare}(\langle T \rangle_c, v, \rho).
\]
PBFT: Normal-case protocol in view $v$

If receive PREPARE message $m$: PREPARE($m$).
PBFT: Normal-case protocol in view $v$

If $n - f$ identical $\text{PREPARE}(m)$ messages: $\text{COMMIT}(m)$. 
If \( n - f \) identical \texttt{Commit}(m) messages: execute, \texttt{Inform}(\langle T \rangle_c, \rho, r).
Theorem

*If the primary is non-faulty and communication is reliable, then the normal-case of PBFT ensures consensus on T in round ρ.*
Theorem

If the primary is non-faulty and communication is reliable, then the normal-case of PBFT ensures consensus on $T$ in round $\rho$.

Example (Byzantine primary, $n = 4$, $f = 1$, $n - f = 3$)
Theorem

*If the primary is non-faulty and communication is reliable, then the normal-case of PBFT ensures consensus on T in round ρ.*

Example (Byzantine primary, \( n = 4, f = 1, n - f = 3 \))
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

*If replicas $R_i, i \in \{1, 2\}$, commit to $m_i = \text{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.*
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

If replicas $R_i$, $i \in \{1, 2\}$, commit to $m_i = \text{PREPARE}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.
Replica $R_i$ commits to $m_i$:

$n - f$ messages $\text{PREPARE}(m_i)$

$R_i$
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

*If replicas $R_i$, $i \in \{1, 2\}$, commit to $m_i = \text{PREPARE}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.*

Proof.
Replica $R_i$ commits to $m_i$:

![Diagram showing replicas and messages]

- $n - f$ messages $\text{PREPARE}(m_i)$
- $\geq n - 2f$ non-faulty
- $\leq f$ faulty

$B_i$ and $F_i$ represent the sets of non-faulty and faulty replicas, respectively.
**PBFT: A normal-case property when\( n > 3f \)**

**Theorem (Castro et al.)**

*If replicas* \( R_i, i \in \{1, 2\} \), *commit to* \( m_i = \text{PrePrepare}(\langle T_i \rangle_{c_i}, \nu, \rho) \), *then* \( \langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2} \).*

**Proof.**

Replica \( R_i \) commits to \( m_i \):

\[
\text{n - f messages PREPARE}(m_i) \\
\geq \text{n - 2f non-faulty} \\
\leq \text{f faulty}
\]

If \( \langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2} \), then \( B_1 \cap B_2 = \emptyset \) and \( |B_1 \cup B_2| \geq 2(n - 2f) \).
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

If replicas $R_i, i \in \{1, 2\}$, commit to $m_i = \text{PREPARE}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica $R_i$ commits to $m_i$:

$n - f$ messages $\text{PREPARE}(m_i)$

$\geq n - 2f$ non-faulty

$\leq f$ faulty

If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(n - 2f)$.

$2(n - 2f) \leq n - f$
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

If replicas $R_i$, $i \in \{1, 2\}$, commit to $m_i = \text{PrePrepare}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.

Proof.

Replica $R_i$ commits to $m_i$:

- $n - f$ messages $\text{Prepare}(m_i)$
- $\geq n - 2f$ non-faulty
- $\leq f$ faulty

If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(n - 2f)$.

$2(n - 2f) \leq n - f \quad \text{iff} \quad 2n - 4f \leq n - f$
PBFT: A normal-case property when $n > 3f$

Theorem (Castro et al.)

*If replicas $R_i, i \in \{1, 2\}$, commit to $m_i = \text{PREPARE}(\langle T_i \rangle_{c_i}, v, \rho)$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.*

Proof.

Replica $R_i$ commits to $m_i$:

- $n - f$ messages $\text{PREPARE}(m_i)$
- $\geq n - 2f$ non-faulty
- $\leq f$ faulty

If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(n - 2f)$.

$$2(n - 2f) \leq n - f \quad \text{iff} \quad 2n - 4f \leq n - f \quad \text{iff} \quad n \leq 3f.$$
PBFT: Primary failure

Primary $P$ is faulty, ignores $R_3$
Primary $P$ is faulty, ignores $R_3$
PBFT: Primary failure

Primary $P$ is faulty, ignores $R_3$

Primary $P$ is non-faulty, $R_3$ pretends to be ignored
**PBFT: Primary failure**

**Primary $P$ is faulty, ignores $R_3$**

**Primary $P$ is non-faulty, $R_3$ pretends to be ignored**
If the primary behaves bad to $> f$ non-faulty replicas, then failure of the primary is detectable.

Replacing the primary: view-change at replica $R$

1. $R$ detects *failure* of the current primary $P$.
2. $R$ chooses a new primary $P'$ (the next replica).
3. $R$ provides $P'$ with its *current state*.
4. $P'$ proposes a *new view*.
5. If the new view is valid, then $R$ switches to this view.
**PbFT: A view-change in view $v$**

Send `VIEWCHANGE(E, v)` with $E$ all prepared transactions.
**PbFT: A view-change in view \( v \)**

If \( n - f \) valid ViewChange\((E, v)\) messages: NewView\((v + 1, E, N)\).

- \( E \) contains \( n - f \) valid ViewChange messages.
- \( N \) contains no-op proposals for *missing rounds.*
**PbFT: A view-change in view $v$**

Move to view $v + 1$ if $\text{NewView}(v + 1, E, N)$ is valid.

- $E$ contains $n - f$ valid $\text{ViewChange}$ messages.
- $N$ contains no-op proposals for *missing rounds*.
PBFT: A property of view-changes when $n > 3f$

Theorem (Castro et al.)

Let $\text{NEWVIEW}(v + 1, \mathcal{E}, N)$ be a well-formed $\text{NEWVIEW}$ message. If a set $S$ of $n - 2f$ non-faulty replicas committed to $m$, then $\mathcal{E}$ contains a $\text{VIEWCHANGE}$ message preparing $m$. 
PBFT: A property of view-changes when \( n > 3f \)

**Theorem (Castro et al.)**

Let \( \text{NEWVIEW}(v + 1, \mathcal{E}, N) \) be a well-formed \( \text{NEWVIEW} \) message. If a set \( S \) of \( n - 2f \) non-faulty replicas committed to \( m \), then \( \mathcal{E} \) contains a \( \text{VIEWCHANGE} \) message preparing \( m \).

**Proof.**

The \( \text{VIEWCHANGE} \) messages in \( \mathcal{E} \):

- \( n - f \) messages \( \text{VIEWCHANGE}(E, v) \)
- \( \geq n - 2f \) non-faulty
- \( \leq f \) faulty

\( B \) contains \( B \cap S \) and \( B \cap F \) is the set of faulty replicas.
Pbft: A property of view-changes when $n > 3f$

Theorem (Castro et al.)

Let $\text{NEWVIEW}(v + 1, E, N)$ be a well-formed $\text{NEWVIEW}$ message. If a set $S$ of $n - 2f$ non-faulty replicas committed to $m$, then $E$ contains a $\text{VIEWCHANGE}$ message preparing $m$.

Proof.

The $\text{VIEWCHANGE}$ messages in $E$:

\begin{align*}
    n - f & \text{ messages } \text{VIEWCHANGE}(E, v) \\
    \geq n - 2f & \text{ non-faulty } \\
    \leq f & \text{ faulty } \\
\end{align*}

if $S \cap B = \emptyset$, then $|S \cup B| \geq 2(n - 2f)$. 
**PBFT: A property of view-changes when \( n > 3f \)**

**Theorem (Castro et al.)**

*Let NewView\((v + 1, E, N)\) be a well-formed NewView message. If a set \( S \) of \( n - 2f \) non-faulty replicas committed to \( m \), then \( E \) contains a ViewChange message preparing \( m \).*

**Proof.**

The ViewChange messages in \( E \):

\[
\begin{align*}
\text{n} - f \text{ messages ViewChange}(E, v) \\
\geq n - 2f \text{ non-faulty} & \quad \rightarrow \quad B \\
\leq f \text{ faulty} & \quad \rightarrow \quad F
\end{align*}
\]

If \( S \cap B = \emptyset \), then \(|S \cup B| \geq 2(n - 2f)|.

\[
2(n - 2f) \leq n - f \quad \text{iiff} \quad 2n - 4f \leq n - f \quad \text{iiff} \quad n \leq 3f.
\]
1. **Undetected failures**: e.g., ignored replicas. At least $n - 2f > f$ non-faulty replicas participate: checkpoints.
PBFT: Further dealing with failures

1. **Undetected failures**: e.g., ignored replicas.
   At least \( n - 2f > f \) non-faulty replicas participate: **checkpoints**.

2. **Detected failures**: primary replacement.
   Worst-case: a sequence of \( f \) view-changes (\( \Omega(f) \) phases).
PbFT: Further dealing with failures

1. **Undetected failures**: e.g., ignored replicas. At least $n - 2f > f$ non-faulty replicas participate: checkpoints.

2. **Detected failures**: primary replacement. Worst-case: a sequence of $f$ view-changes ($\Omega (f)$ phases).

3. **View-change cost**: includes all previous transactions! Checkpoints: view-change includes last successful checkpoint.
1. **Undetected failures**: e.g., ignored replicas. At least \( n - 2f > f \) non-faulty replicas participate: *checkpoints*.

2. **Detected failures**: primary replacement. Worst-case: a sequence of \( f \) view-changes (\( \Omega (f) \) phases).

3. **View-change cost**: includes all previous transactions! Checkpoints: view-change includes *last successful* checkpoint.

4. **Unreliable communication**: replacement of good primaries. Worst-case: replacements until communication becomes *reliable*. 
### Other consensus protocols: Go beyond PBFT

<table>
<thead>
<tr>
<th>Feature</th>
<th>PBFT</th>
<th>Zyzzyva</th>
<th>HotStuff</th>
<th>ALGORAND</th>
<th>RBFT</th>
<th>SynBFT</th>
<th>CheaBFT</th>
<th>PoE</th>
<th>GeoBFT</th>
<th>MultiBFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous communication</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using authenticated channels</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-round reasoning</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speculative execution</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Randomized primary election</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold signatures</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improved reliability</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous primary replacement</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Per-round checkpoints</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trusted components</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using sub-quorums</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geo-scale clustering</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consensus parallelization</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The cluster-sending problem
Vision: Resilient high-performance data processing

Europe

Malicious

America

Crashed

Requirement for geo-scale aware sharding
Fault-tolerant communication between Byzantine clusters!
The need for cluster-sending

Definition
The *cluster-sending problem* is the problem of sending a value \( v \) from \( C_1 \) to \( C_2 \) such that:

1. all non-faulty replicas in \( C_2 \) *receive* the value \( v \);
2. only if all non-faulty replicas in \( C_1 \) *agree* upon sending the value \( v \) to \( C_2 \) will non-faulty replicas in \( C_2 \) receive \( v \);
3. all non-faulty replicas in \( C_1 \) can *confirm* that the value \( v \) was received.

Straightforward cluster-sending solution (crash failures)
Pair-wise broadcasting with \((f_1 + 1)(f_2 + 1) \approx f_1 \times f_2\) messages.
Global versus local communication

Straightforward cluster-sending solution (crash failures)
Pair-wise broadcasting with \((f_1 + 1)(f_2 + 1) \approx f_1 \times f_2\) messages.

<table>
<thead>
<tr>
<th></th>
<th>Ping round-trip times (ms)</th>
<th>Bandwidth (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OR  IA  Mont.  BE  TW  Syd.</td>
<td>OR  IA  Mont.  BE  TW  Syd.</td>
</tr>
<tr>
<td>Oregon</td>
<td>(\leq 1) 38 65 136 118 161</td>
<td>7998 669 371 194 188 136</td>
</tr>
<tr>
<td>Iowa</td>
<td>(\leq 1) 33 98 153 172</td>
<td>10004 752 243 144 120</td>
</tr>
<tr>
<td>Montreal</td>
<td>(\leq 1) 82 186 202</td>
<td>7977 283 111 102</td>
</tr>
<tr>
<td>Belgium</td>
<td>(\leq 1) 252 270</td>
<td>9728 79 66</td>
</tr>
<tr>
<td>Taiwan</td>
<td>(\leq 1) 137</td>
<td>7998 160</td>
</tr>
<tr>
<td>Sydney</td>
<td>(\leq 1)</td>
<td>7977</td>
</tr>
</tbody>
</table>
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \]
\[ n_2 = 5 \]
\[ f_1 = 7 \]
\[ f_2 = 2 \]

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)
Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.
Lower bounds for cluster-sending: Example

\[ n_1 = 15 \quad f_1 = 7 \]
\[ n_2 = 5 \quad f_2 = 2 \]

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.
Theorem (Cluster-sending lower bound, crash failures)

Assume $n_1 \geq n_2$ and let

\[
q = (f_1 + 1) \text{div } nf_2;
\]
\[
r = (f_1 + 1) \text{mod } nf_2;
\]
\[
\sigma = qn_2 + r + f_2 \text{ sgn } r.
\]

We need to exchange at least $\sigma$ messages to do cluster-sending.

- Similar results for $n_1 \leq n_2$.
- If $n_1 \approx n_2$: at least $f_1 + f_2 + 1$ messages.
Cluster-sending with Byzantine failures

Theorem (Cluster-sending lower bound, Byzantine failures)
Assume \( n_1 \geq n_2 \) and let

\[
q = (2f_1 + 1) \div nf_2;
\]
\[
r = (f_1 + 1) \mod nf_2;
\]
\[
\sigma = qn_2 + r + f_2 \text{ sgn } r.
\]

We need to exchange at least \( \sigma \) digital signatures to do cluster-sending.

- Similar results for \( n_1 \leq n_2 \).
- If \( n_1 \approx n_2 \): at least \( 2f_1 + f_2 + 1 \) digital signatures.
- Only authenticated communication: much harder!
An optimal cluster-sending algorithm (crash failures)

Protocol for the sending cluster $C_1$, $n_1 \geq n_2$, $n_1 \geq \sigma$:

1: Choose replicas $\mathcal{P} \subseteq C_1$ with $|\mathcal{P}| = \sigma$.
2: Choose a $n_2$-partition $\text{partition}(\mathcal{P})$ of $\mathcal{P}$.
3: **for** $P \in \text{partition}(\mathcal{P})$ **do**
4: Choose replicas $Q \subseteq C_2$ with $|Q| = |P|$.
5: Choose a bijection $b : P \rightarrow Q$.
6: **for** $R_1 \in P$ **do**
7: Send $v$ from $R_1$ to $b(R_1)$.

Protocol for the receiving cluster $C_2$:

8: **event** $R_2 \in C_2$ receives $w$ from a replica in $C_1$ **do**
9: Broadcast $w$ to all replicas in $C_2$.
10: **event** $R'_2 \in C_2$ receives $w$ from a replica in $C_2$ **do**
11: $R'_2$ considers $w$ received.
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = n_2 = 4$, $f_1 = f_2 = 1$, $\sigma = 3$

<table>
<thead>
<tr>
<th>$R_{2,1}$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{2,2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{2,3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{2,4}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$C_2$

<table>
<thead>
<tr>
<th>$R_{1,1}$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{1,2}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{1,3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_{1,4}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$C_1$

Decide on sending $v$
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = n_2 = 4$, $f_1 = f_2 = 1$, $\sigma = 3$

Decide on sending $v$
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = n_2 = 4$, $f_1 = f_2 = 1$, $\sigma = 3$

Decide on sending $v$

Received $v$

Similar algorithm can deal with Byzantine failures ($\sigma = 4$).
An optimal cluster-sending algorithm—visualized

Crash failures, $n_1 = n_2 = 4$, $f_1 = f_2 = 1$, $\sigma = 3$

$C_2 \{ R_{2,1}, R_{2,2}, R_{2,3}, R_{2,4} \}$

$C_1 \{ R_{1,1}, R_{1,2}, R_{1,3}, R_{1,4} \}$

Decide on sending $\nu$

Received $\nu$

Similar algorithm can deal with Byzantine failures ($\sigma = 4$).
Crash failures, $n_1 = n_2 = 4$, $f_1 = f_2 = 1$, $\sigma = 3$

Similar algorithm can deal with Byzantine failures ($\sigma = 4$).
Conclusion

Efficient cluster-sending is possible.

Ongoing work: Initial results

The Byzantine learner problem
Vision: Specializing for read-only workloads

Updates (e.g., write transactions) → Malicious

Analytics
Data Provenance
Machine Learning
Visualization

Read-only workloads

Requirement for data-hungry read-only workloads
Stream all data updates with low cost for all replicas involved.
Vision: Specializing for read-only workloads

Requirement for data-hungry read-only workloads
Stream all data updates with low cost for all replicas involved.

Cluster-sending? Optimal for single messages, not for streams!
The need for Byzantine learning

Definition
Let $C$ be a cluster deciding on a sequence of transactions. The *Byzantine learning problem* is the problem of sending the decided transactions from $C$ to a learner $L$ such that:

- the $L$ will eventually *receive all* decided transactions;
- the $L$ will *only receive* decided transactions.
The need for Byzantine learning

Definition
Let $C$ be a cluster deciding on a sequence of transactions. The *Byzantine learning problem* is the problem of sending the decided transactions from $C$ to a learner $L$ such that:

- the $L$ will eventually *receive all* decided transactions;
- the $L$ will *only receive* decided transactions.

Practical requirements

- Minimizing overall communication.
- Load balancing among all replicas in $C$. 
Background: Information dispersal algorithms

Definition
Let $v$ be a value with storage size $\|v\|$.
An \textit{information dispersal algorithm} can encode $v$ in $n$ pieces $v'$ such that $v$ can be \textit{decoded} from every set of $n - f$ such pieces.

The algorithm is \textit{optimal} if each piece $v'$ has size $\lceil \|v\|/(n - f) \rceil$.
In this case, the $n - f$ pieces necessary for decoding have total size:

$$ (n - f) \left\lceil \frac{\|v\|}{(n - f)} \right\rceil \approx \|v\|. $$

Theorem (Rabin)
\textit{The IDA information dispersal algorithm is optimal.}
The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$
The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$

1. Partition the Blockchain in sequences $S$ of $n$ transactions.

Observations ($n > 2f$)

- Each sequence $S$ has size $\|S\| = \Omega(n)$.
- Each piece $S_i$ has size $\|S_i\| = \lceil \|S\|/(n-f) \rceil$.
- Learner $L$ receives at most $B = n(\lceil \|S\|/(n-f) \rceil + c)$ bytes:
  
  $B \leq n(\|S\|/n - f + 1 + c) < 2\|S\| + n + nc = O(\|S\| + cn)$. 

The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$

1. Partition the Blockchain in sequences $S$ of $n$ transactions.
2. Replica $R_i \in C$ encodes $S$ into the $i$-th IDA piece $S_i$.

Observations ($n > 2f$)

▶ Each sequence $S$ has size $\|S\| = \Omega(n)$.
▶ Each piece $S_i$ has size $\|S_i\| = \lceil \|S\| / (n - f) \rceil$.
▶ Learner $L$ receives at most $B = n (\lceil \|S\| / (n - f) \rceil + c)$ bytes:
  $$B \leq n (\|S\| n - f + 1 + c) < 2 \|S\| + n + nc = O(\|S\| + cn)$$
The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$

1. Partition the Blockchain in sequences $S$ of $n$ transactions.
2. Replica $R_i \in C$ encodes $S$ into the $i$-th IDA piece $S_i$.
3. Replica $R_i \in C$ sends $S_i$ with a checksum $C_i(S)$ of $S$ to $L$. 

Observations ($n > 2f$)

- Each sequence $S$ has size $\|S\| = \Omega(n)$.
- Each piece $S_i$ has size $\|S_i\| = \lceil \|S\|/(n-f) \rceil$.
- Learner $L$ receives at most $B = n(\|S\|/(n-f)+1+c)$ bytes:
  $$B \leq n(\|S\|/(n-f)+1+c) \leq 2\|S\|+n+n_c = O(\|S\|+cn)$$
The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$

1. Partition the Blockchain in sequences $S$ of $n$ transactions.
2. Replica $R_i \in C$ encodes $S$ into the $i$-th IDA piece $S_i$.
3. Replica $R_i \in C$ sends $S_i$ with a checksum $C_i(S)$ of $S$ to $L$.
4. $L$ receives at least $n - f$ distinct pieces and decodes $S$. 
The delayed-replication algorithm

Idea: $C$ sends a Blockchain to learner $L$

1. Partition the Blockchain in sequences $S$ of $n$ transactions.
2. Replica $R_i \in C$ encodes $S$ into the $i$-th IDA piece $S_i$.
3. Replica $R_i \in C$ sends $S_i$ with a checksum $C_i(S)$ of $S$ to $L$.
4. $L$ receives at least $n - f$ distinct pieces and decodes $S$.

Observations ($n > 2f$)

- Each sequence $S$ has size $\|S\| = \Omega(n)$.
- Each piece $S_i$ has size $\|S_i\| = \lceil \|S\|/(n - f) \rceil$.
- Learner $L$ receives at most $B = n(\lceil \|S\|/(n - f) \rceil + c)$ bytes:

\[
B \leq n \left( \frac{\|S\|}{n - f} + 1 + c \right) < 2\|S\| + n + nc = O(\|S\| + cn).
\]
Communication by the delayed-replication algorithm

Consensus decisions (transactions) \[\rightarrow\]

- No dispersal
- First 4 transactions
- Second 4 transactions
Decoding $S$ using simple checksums ($n > 2f$)

- Use checksums $\text{hash}(S)$.
- The $n - f$ non-faulty replicas will provide correct pieces.
- At least $n - f > f$ messages with correct checksums.
- Received some forged pieces?
  - Decoding yields $S'$.
  - $\text{hash}(S') \neq \text{hash}(S)$.
  - Use other pieces.
- Compute intensive for learner.
Decoding $S$ using tree checksums

Use Merkle-trees to construct checksums
Consider 8 replicas and a sequence $S$.
We construct the checksum $C_5(S)$ of $S$ (used by $R_5$).

Construct a Merkle tree for pieces $S_0, \ldots, S_7$. 
Use Merkle-trees to construct checksums
Consider 8 replicas and a sequence \( S \).
We construct the checksum \( C_5(S) \) of \( S \) (used by \( R_5 \)).

Determine the path from root to \( S_5 \).
Decoding $S$ using tree checksums

Use Merkle-trees to construct checksums

Consider 8 replicas and a sequence $S$.

We construct the checksum $C_5(S)$ of $S$ (used by $R_5$).

Select root and neighbors: $C_5(S) = [h_4, h_{67}, h_{0123}, h_{01234567}]$. 
Decoding $S$ using tree checksums

Use Merkle-trees to construct checksums
Consider 8 replicas and a sequence $S$.
We construct the checksum $C_5(S)$ of $S$ (used by $R_5$).

Enables recognizing forged pieces before decoding.
Delayed-replication: Main result \((n > 2f)\)

**Theorem**

Consider the learner \(L\), replica \(R\), and decided transactions \(\mathcal{T}\). The delayed-replication algorithm with tree checksums guarantees

1. \(L\) will learn \(\mathcal{T}\);
2. \(L\) will receive at most \(|\mathcal{T}|\) messages with a total size of
   \[
   O\left(\|\mathcal{T}\| \left(\frac{n}{n-f} \right) + |\mathcal{T}| \log n\right) = O\left(\|\mathcal{T}\| + |\mathcal{T}| \log n\right);
   \]
3. \(L\) will only need at most \(|\mathcal{T}|/n\) decode steps;
4. \(R\) will sent at most \(|\mathcal{T}|/n\) messages to \(L\) of size
   \[
   O\left(\frac{\|\mathcal{T}\|}{n-f} + \frac{|\mathcal{T}| \log n}{n}\right) = O\left(\frac{\|\mathcal{T}\| + |\mathcal{T}| \log n}{n}\right).
   \]
Conclusion

Efficient Byzantine learning is possible.

Blockchain applications

- Low-cost checkpoint protocols.
- Scalable storage for resilient systems.

Ongoing work: Initial results

About us

- Jelle Hellings
  https://jhellings.nl/

- Expolab
  https://expolab.org/

- ResilientDB
  https://resilientdb.com/
References I


References III


References IV


References V


References VI


