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

Suyash Gupta
Jelle Hellings
Sajjad Rahnama
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.
6. Cryptography.

Main focus today

Permissioned, Byzantine Fault tolerance, Asynchronous.

Membership: Permissionless versus permissioned

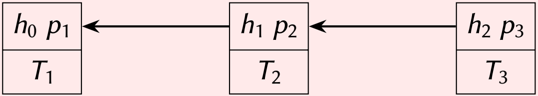
Permissionless	Participants are not known. Can provide *open membership*.
Permissioned	Participants are known and vetted.

Permissionless	Permissioned
Public Blockchains	Traditional resilient systems (PBFT, ...)
Bitcoin	ResilientDB
Ethereum	HyperLedger
...	...

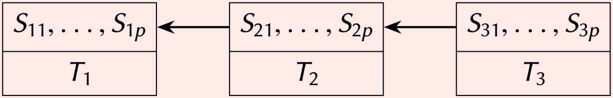
Membership: Tamper-proof structures

How is the Blockchain made tamper-proof?

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



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



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

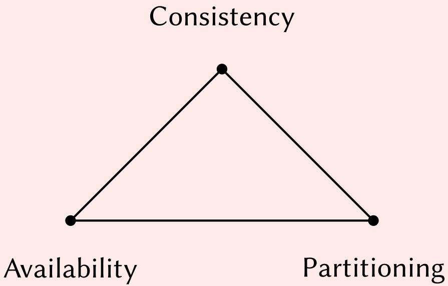
Consistency	Does every participant have exactly the same data?
Availability	Does the system continuously provide services?
Partitioning	Can the system cope with network disturbances?

Theorem (The CAP Theorem)

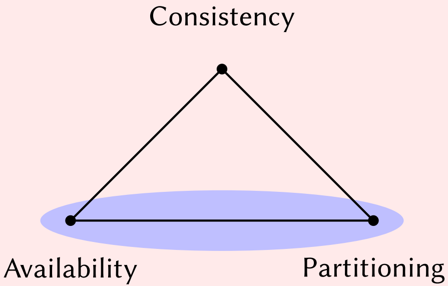
Can provide at most two-out-of-three of these properties.

CAP Theorem uses narrow definitions!

The CAP Theorem and Blockchains



The CAP Theorem and Blockchains

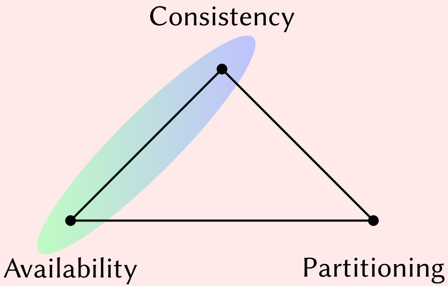


Permissionless Blockchains

Open membership focuses on Availability and Partitioning.

$\implies$ Consistency not guaranteed (e.g., forks).

The CAP Theorem and Blockchains

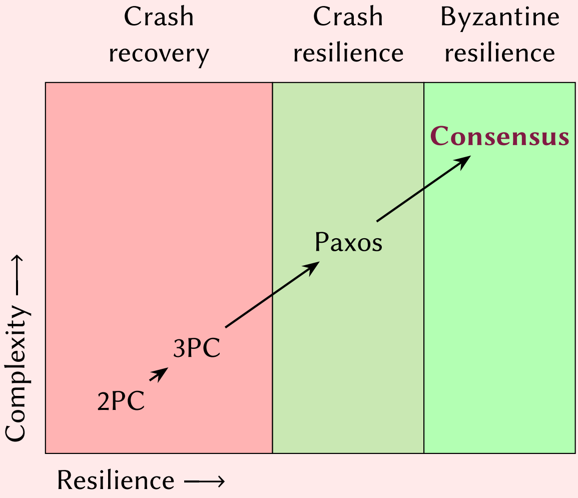


Permissioned Blockchains

Consistency at all costs.

$\implies$ 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.

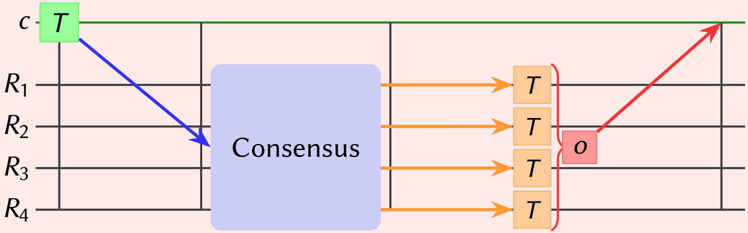
Consensus in permissioned Blockchains

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.



From consensus to a consistent Blockchain

Reminder: append-only sequence of transactions.

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

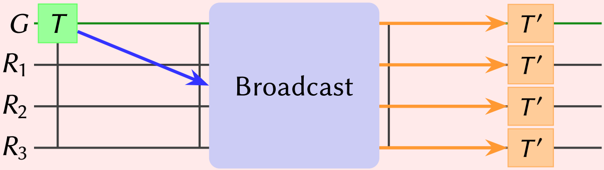
Consistent state: linearizable order and deterministic execution

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.



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

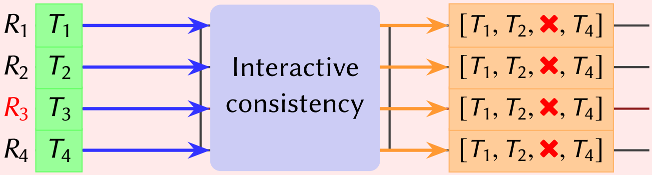
Interactive consistency

Assume **n** replicas and each replica R_i holds a transaction T_i.

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.



(As R_3 is faulty: ✖ 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.

Permissionless	Permissioned
Cryptographic primitives	Cryptographic primitives
Majority of resources	Majority of participants

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 $\mathbf{n}$ *replicas of which at most* $\mathbf{f}$ *are Byzantine.*

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

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 $\mathbf{n}$ *replicas of which at most* $\mathbf{f}$ *are Byzantine.*

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

Bounds for consensus: response via majority votes

For clients to learn outcome, we require at least $\mathbf{n} > 2\mathbf{f}$.

Lower bounds on communication (phases, messages)

Theorem (Fisher and Lynch; Dolev, Reischuk, and Strong)

Assume a system with $\mathbf{n}$ *replicas of which at most* $\mathbf{f}$ *can be Byzantine.*

1. *Consensus: worst-case* $\Omega(\mathbf{f} + 1)$ *phases of communication.*
2. *Optimistic Broadcasts:* $\Omega(t + 2)$ *phases if* $t \leq \mathbf{f}$ *failures happen.*

Lower bounds on communication (phases, messages)

Theorem (Fisher and Lynch; Dolev, Reischuk, and Strong)

Assume a system with $\mathbf{n}$ *replicas of which at most* $\mathbf{f}$ *can be Byzantine.*

1. *Consensus: worst-case* $\Omega(\mathbf{f}+1)$ *phases of communication.*
2. *Optimistic Broadcasts:* $\Omega(t+2)$ *phases if* $t \leq \mathbf{f}$ *failures happen.*

Theorem (Dolev and Reischuk)

Assume a system with $\mathbf{n}$ *replicas of which at most* $\mathbf{f}$ *can be Byzantine.*
Any broadcast protocol using digital signatures requires:

1. $\Omega(\mathbf{nf})$ *digital signatures;*
2. $\Omega\left(\mathbf{n}+\mathbf{f}^2\right)$ *messages.*

Connectivity of the replicas and quality of the network

Theorem (Dolev)

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

Connectivity of the replicas and quality of the network

Theorem (Dolev)

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

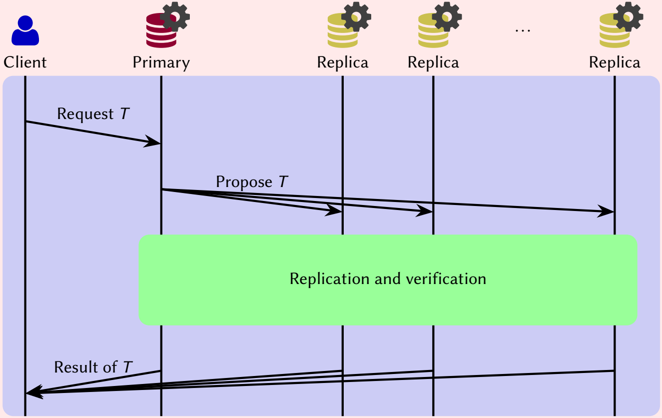
- Dealing with $\mathbf{f}$ malicious failures requires $\mathbf{n} > 3\mathbf{f}$ replicas.
- Worst-case: at least $\Omega(\mathbf{f} + 1)$ phases of communication.
- Worst-case: at least $\Omega(\mathbf{nf})$ signatures and $\Omega\left(\mathbf{n} + \mathbf{f}^2\right)$ messages.
- Termination: reliable communication
  - Between most replicas;
  - Communication with bounded-delay.

A practical consensus protocol: PBFT

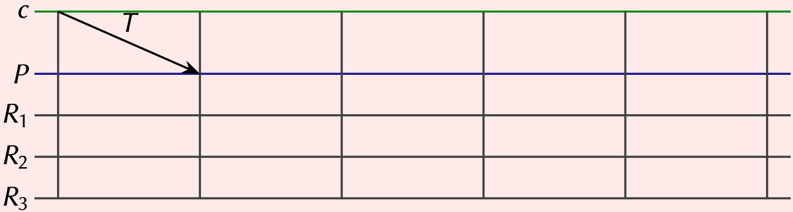
PBFT: Practical Byzantine Fault Tolerance

Primary Coordinates consensus: propose transactions to replicate.

Backup Accept transactions and verifies behavior of primary.

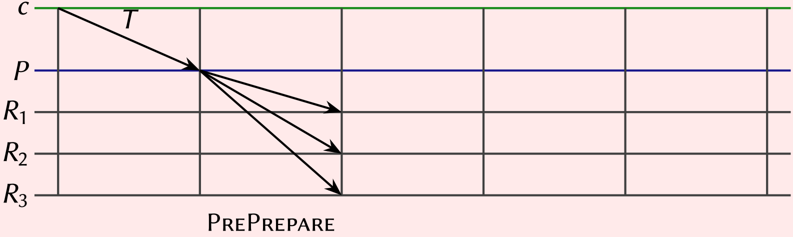


PBFT: Normal-case protocol in view v



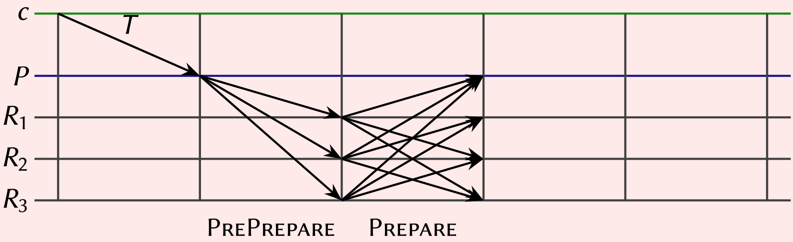
$$\langle T \rangle_c.$$

PBFT: Normal-case protocol in view v



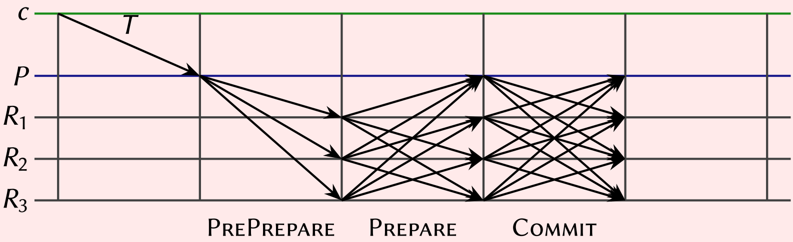
$\text{PrePrepare}(\langle T \rangle_c, v, \rho)$.

PBFT: Normal-case protocol in view v



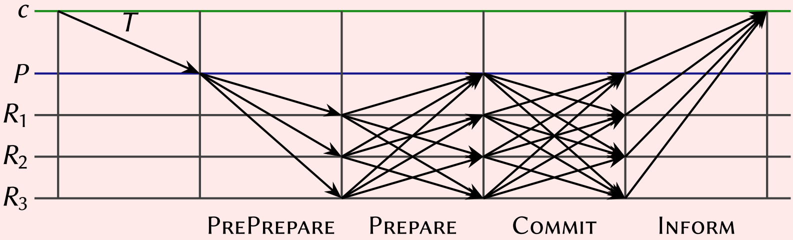
If receive PREPREPARE message m: PREPARE(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical PREPARE(m) messages: COMMIT(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical COMMIT(m) messages: execute, INFORM($\langle T \rangle_c, \rho, r$).

PBFT: Normal-case consensus

Theorem

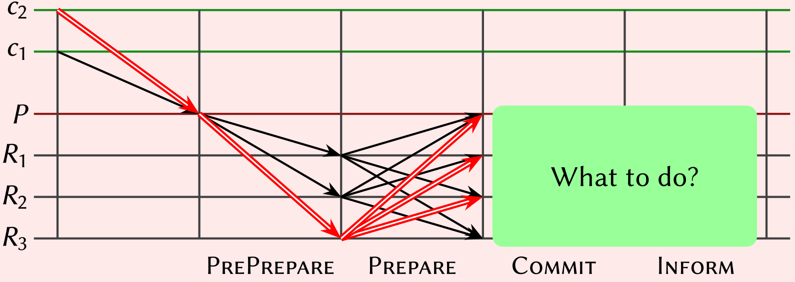
If the primary is non-faulty and communication is reliable, then the normal-case of PBFT *ensures consensus on T in round ρ.*

PBFT: Normal-case consensus

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, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)

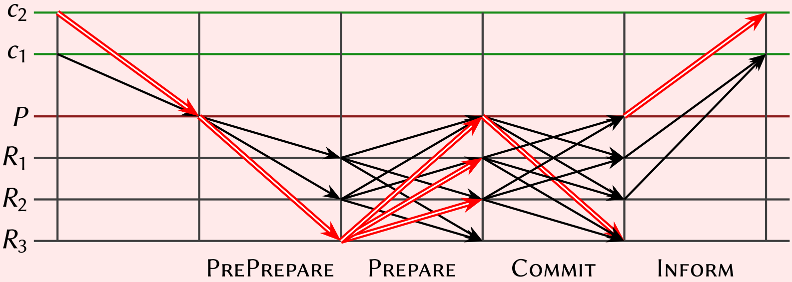


PBFT: Normal-case consensus

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, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)



PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

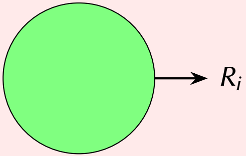
Theorem (Castro et al.)

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

Proof.

Replica R_i commits to m_i:

$\mathbf{n} - \mathbf{f}$ messages $\textsc{Prepare}(m_i)$



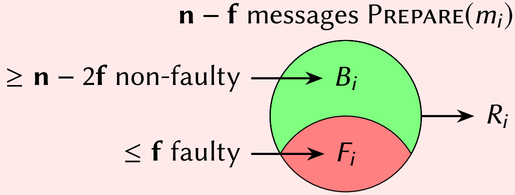
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

Replica R_i commits to m_i:



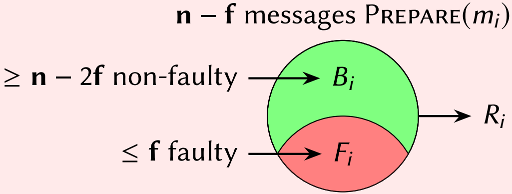
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

Replica R_i commits to m_i:



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

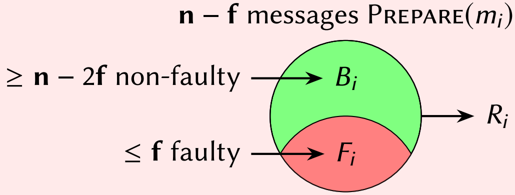
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

Replica R_i commits to m_i:



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

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f}$$

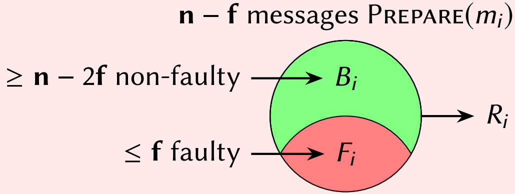
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

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:



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

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f} \qquad \text{iff} \qquad 2\mathbf{n} - 4\mathbf{f} \leq \mathbf{n} - \mathbf{f}$$

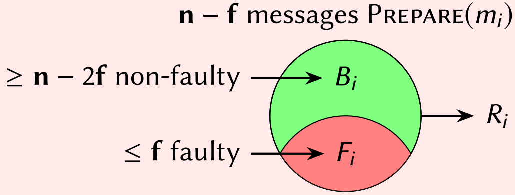
PBFT: A normal-case property when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

Replica R_i commits to m_i:

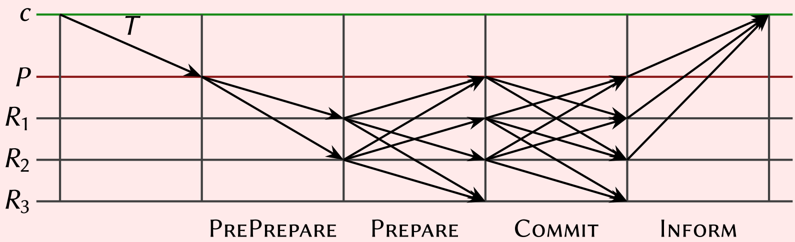


If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then $B_1 \cap B_2 = \emptyset$ and $|B_1 \cup B_2| \geq 2(\mathbf{n} - 2\mathbf{f})$.

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f} \quad \text{iff} \quad 2\mathbf{n} - 4\mathbf{f} \leq \mathbf{n} - \mathbf{f} \quad \text{iff} \quad \mathbf{n} \leq 3\mathbf{f}.$$

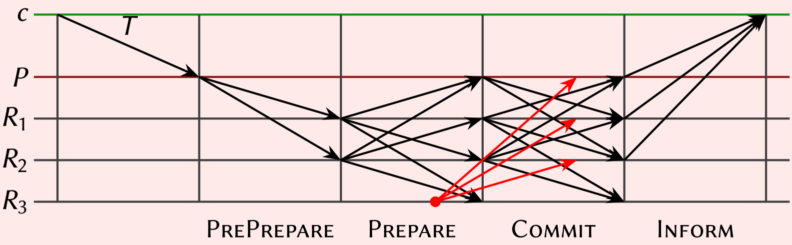
PBFT: Primary failure

Primary P is faulty, ignores R_3



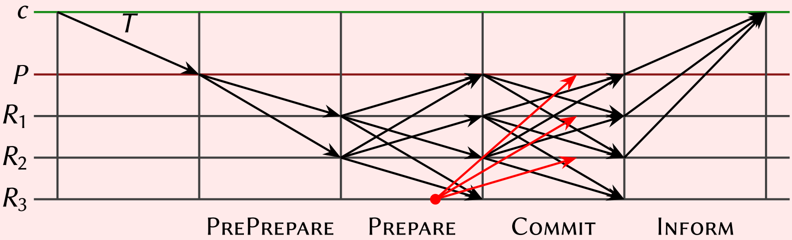
PBFT: Primary failure

Primary P is faulty, ignores R_3

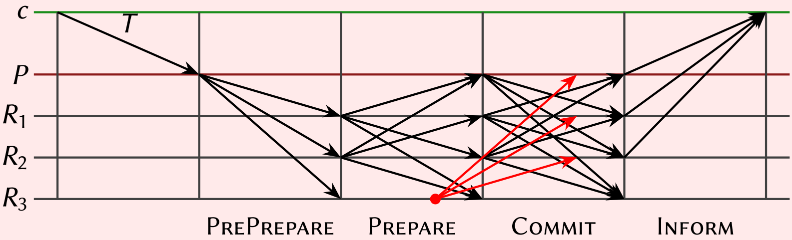


PBFT: Primary failure

Primary P is faulty, ignores R_3

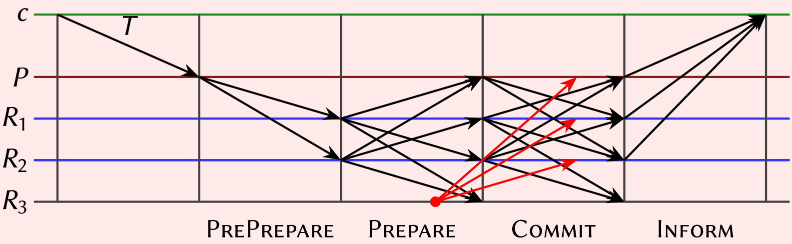


Primary P is non-faulty, R_3 pretends to be ignored

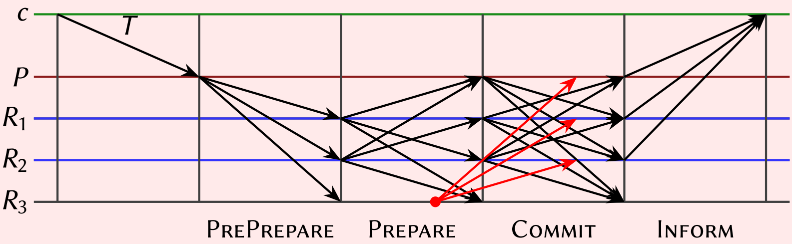


PBFT: Primary failure

Primary P is faulty, ignores R_3



Primary P is non-faulty, R_3 pretends to be ignored



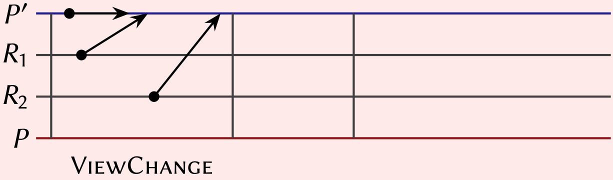
PBFT: Detectable primary failures

If the primary behaves bad to > $\mathbf{f}$ non-faulty replicas,
then failure of the primary is detectable.

Replacing the primary: view-change at replica R

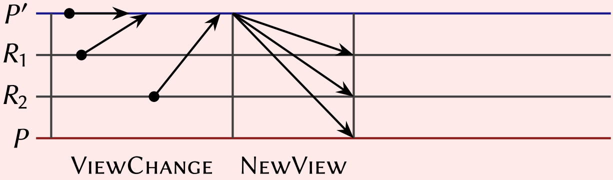
1. R detects *failure* of the current primary P.
2. R chooses a new primary P' (the next replica).
3. R provides P' with its *current state*.
4. P' proposes a *new view*.
5. If the new view is valid, then R switches to this view.

PBFT: A view-change in view v



Send VIEWCHANGE(E, v) with E all prepared transactions.

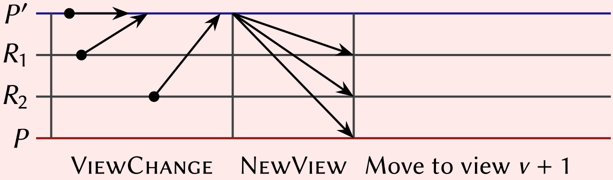
PBFT: A view-change in view v



If $\mathbf{n} - \mathbf{f}$ valid VIEWCHANGE(E, v) messages: NEWVIEW$(v + 1, \mathcal{E}, \mathcal{N})$.

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid VIEWCHANGE messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

PBFT: A view-change in view v



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

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid ViewChange messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

PBFT: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

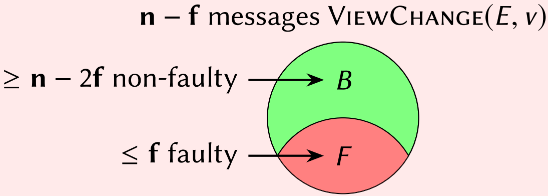
Pbft: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

The ViewChange messages in $\mathcal{E}$:



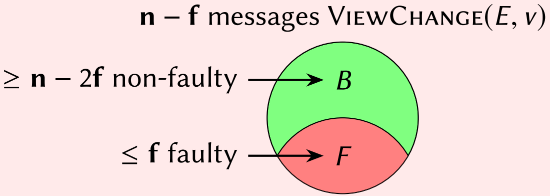
PBFT: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

Theorem (Castro et al.)

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

Proof.

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



if $S \cap B = \emptyset$, then $|S \cup B| \geq 2(\mathbf{n} - 2\mathbf{f})$.

PBFT: A property of view-changes when $\mathbf{n} > 3\mathbf{f}$

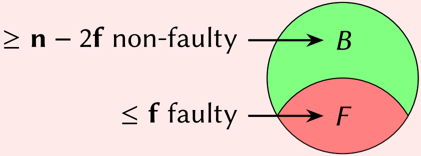
Theorem (Castro et al.)

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

Proof.

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

$\mathbf{n} - \mathbf{f}$ messages VIEWCHANGE(E, v)



if $S \cap B = \emptyset$, then $|S \cup B| \geq 2(\mathbf{n} - 2\mathbf{f})$.

$$2(\mathbf{n} - 2\mathbf{f}) \leq \mathbf{n} - \mathbf{f} \quad \text{iff} \quad 2\mathbf{n} - 4\mathbf{f} \leq \mathbf{n} - \mathbf{f} \quad \text{iff} \quad \mathbf{n} \leq 3\mathbf{f}.$$

PBFT: Further dealing with failures

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

PBFT: Further dealing with failures

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

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

PBFT: Further dealing with failures

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

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

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

PBFT: Further dealing with failures

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

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

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

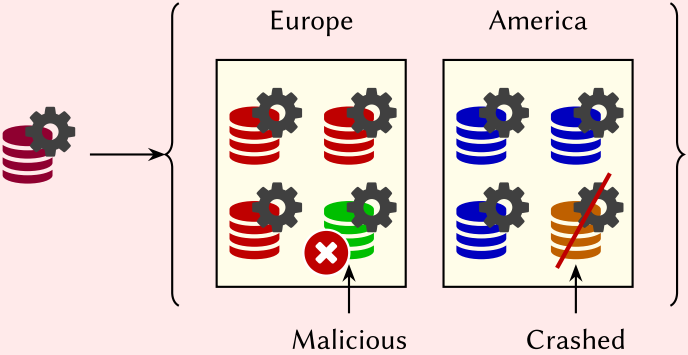
4. *Unreliable communication*: replacement of good primaries.
   Worst-case: replacements until communication becomes *reliable*.

Other consensus protocols: Go beyond PBFT

	PBFT	ZYZZYVA	HOTSTUFF	ALGORAND	RBFT	SYNBFT	CHEAPBFT	POE	GEOBFT	MULTIBFT
Synchronous communication						✓				
Using authenticated channels	✓							✓		
Multi-round reasoning			✓					✓		
Speculative execution		✓						✓		
Randomized primary election				✓		✓				
Threshold signatures			✓							
Improved reliability					✓					✓
Continuous primary replacement			✓							
Per-round checkpoints			✓							
Trusted components							✓			
Using sub-quorums				✓					✓	
Geo-scale clustering									✓	
Consensus parallelization									✓	✓

The cluster-sending problem

Vision: Resilient high-performance data processing



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 $\mathcal{C}_1$ to $\mathcal{C}_2$ such that:

1. all non-faulty replicas in $\mathcal{C}_2$ *receive* the value v;
2. only if all non-faulty replicas in $\mathcal{C}_1$ *agree* upon sending the value v to $\mathcal{C}_2$ will non-faulty replicas in $\mathcal{C}_2$ receive v;
3. all non-faulty replicas in $\mathcal{C}_1$ can *confirm* that the value v was received.

Straightforward cluster-sending solution (crash failures)

Pair-wise broadcasting with $(\mathbf{f}_1 + 1)(\mathbf{f}_2 + 1) \approx \mathbf{f}_1 \times \mathbf{f}_2$ messages.

Global versus local communication

Straightforward cluster-sending solution (crash failures)

Pair-wise broadcasting with $(\mathbf{f}_1 + 1)(\mathbf{f}_2 + 1) \approx \mathbf{f}_1 \times \mathbf{f}_2$ messages.

	Ping round-trip times (ms)						*Bandwidth (Mbit/s)*					
	OR	IA	Mont.	BE	TW	Syd.	OR	IA	Mont.	BE	TW	Syd.
Oregon	≤ 1	38	65	136	118	161	7998	669	371	194	188	136
Iowa		≤ 1	33	98	153	172		10004	752	243	144	120
Montreal			≤ 1	82	186	202			7977	283	111	102
Belgium				≤ 1	252	270				9728	79	66
Taiwan					≤ 1	137					7998	160
Sydney						≤ 1						7977

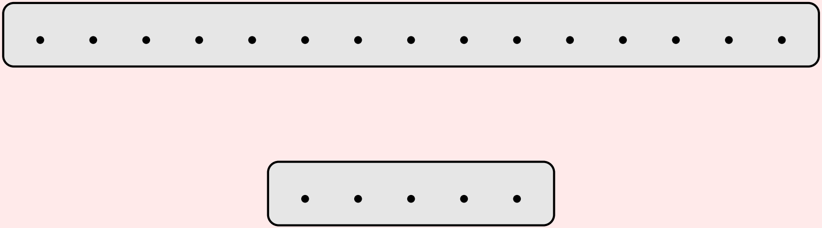
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



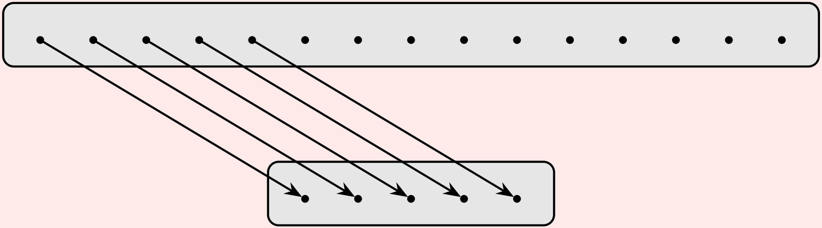
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\quad$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\quad$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



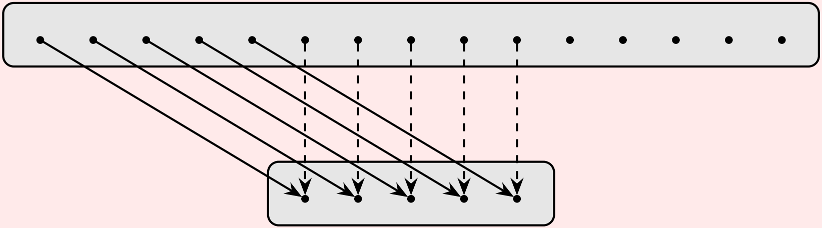
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



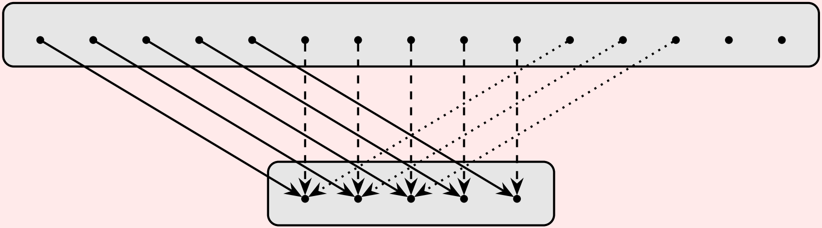
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



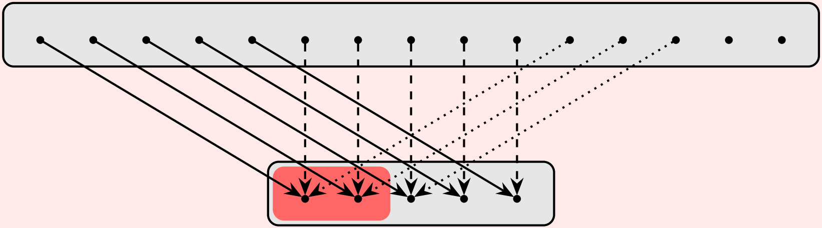
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\qquad$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\qquad$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



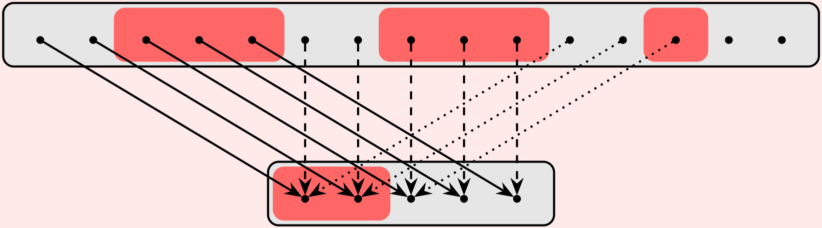
Lower bounds for cluster-sending: Example

$\mathbf{n}_1 = 15$ $\mathbf{f}_1 = 7$

$\mathbf{n}_2 = 5$ $\mathbf{f}_2 = 2$

Claim (crash failures)

Any correct protocol needs to send at least 14 messages.



Lower bounds for cluster-sending: Results

Theorem (Cluster-sending lower bound, crash failures)

Assume $\mathbf{n}_1 \geq \mathbf{n}_2$ *and let*

$$
\begin{aligned}
q &= (\mathbf{f}_1 + 1) \operatorname{div} \mathbf{nf}_2;\\
r &= (\mathbf{f}_1 + 1) \bmod \mathbf{nf}_2;\\
\sigma &= q\mathbf{n}_2 + r + \mathbf{f}_2 \operatorname{sgn} r.
\end{aligned}
$$

We need to exchange at least σ *messages to do cluster-sending.*

- Similar results for $\mathbf{n}_1 \leq \mathbf{n}_2$.
- If $\mathbf{n}_1 \approx \mathbf{n}_2$: at least $\mathbf{f}_1 + \mathbf{f}_2 + 1$ messages.

Cluster-sending with Byzantine failures

Theorem (Cluster-sending lower bound, Byzantine failures)

Assume $\mathbf{n}_1 \geq \mathbf{n}_2$ and let

$$q = (\mathbf{2f}_1 + 1) \operatorname{div} \mathbf{nf}_2;$$
$$r = (\mathbf{f}_1 + 1) \operatorname{mod} \mathbf{nf}_2;$$
$$\sigma = q\mathbf{n}_2 + r + \mathbf{f}_2 \operatorname{sgn} r.$$

We need to exchange at least σ digital signatures to do cluster-sending.

- Similar results for $\mathbf{n}_1 \leq \mathbf{n}_2$.
- If $\mathbf{n}_1 \approx \mathbf{n}_2$: at least $\mathbf{2f}_1 + \mathbf{f}_2 + 1$ digital signatures.
- Only authenticated communication: much harder!

An optimal cluster-sending algorithm (crash failures)

Protocol for the sending cluster $\mathcal{C}_1$, $\mathbf{n}_1 \geq \mathbf{n}_2$, $\mathbf{n}_1 \geq \sigma$:

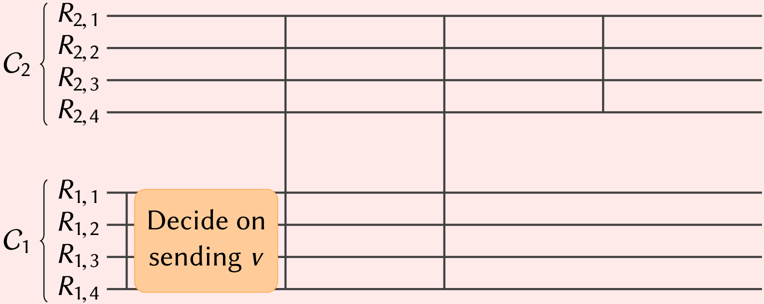
```
Choose replicas 𝒫 ⊆ C₁ with |𝒫| = σ.
Choose a n₂-partition partition(𝒫) of 𝒫.
for P ∈ partition(𝒫) do
    Choose replicas Q ⊆ C₂ with |Q| = |P|.
    Choose a bijection b : P → Q.
    for R₁ ∈ P do
        Send v from R₁ to b(R₁).
```

Protocol for the receiving cluster $\mathcal{C}_2$:

```
event R₂ ∈ C₂ receives w from a replica in C₁ do
    Broadcast w to all replicas in C₂.
event R₂′ ∈ C₂ receives w from a replica in C₂ do
    R₂′ considers w received.
```

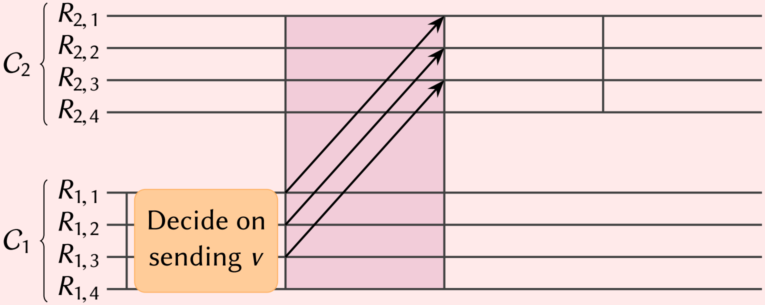
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = \mathbf{n}_2 = 4$, $\mathbf{f}_1 = \mathbf{f}_2 = 1$, $\sigma = 3$



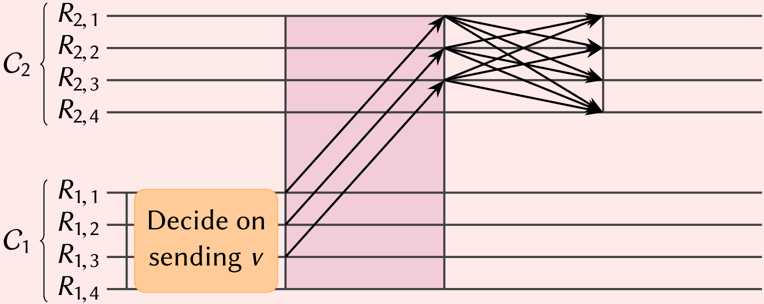
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = \mathbf{n}_2 = 4$, $\mathbf{f}_1 = \mathbf{f}_2 = 1$, $\sigma = 3$



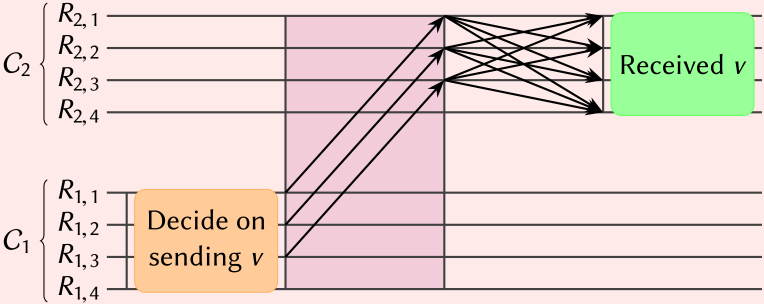
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = \mathbf{n}_2 = 4$, $\mathbf{f}_1 = \mathbf{f}_2 = 1$, $\sigma = 3$



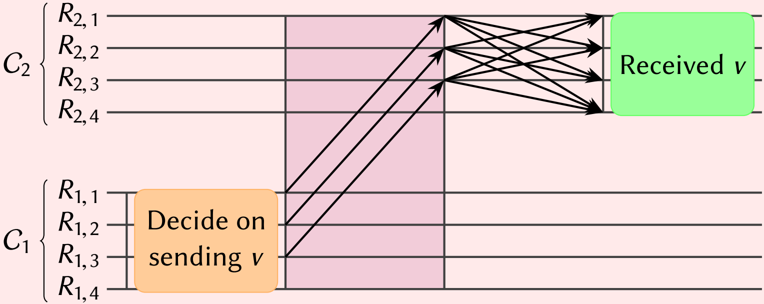
An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = \mathbf{n}_2 = 4$, $\mathbf{f}_1 = \mathbf{f}_2 = 1$, $\sigma = 3$



An optimal cluster-sending algorithm—visualized

Crash failures, $\mathbf{n}_1 = \mathbf{n}_2 = 4$, $\mathbf{f}_1 = \mathbf{f}_2 = 1$, $\sigma = 3$



Similar algorithm can deal with Byzantine failures ($\sigma = \mathbf{4}$).

Conclusion

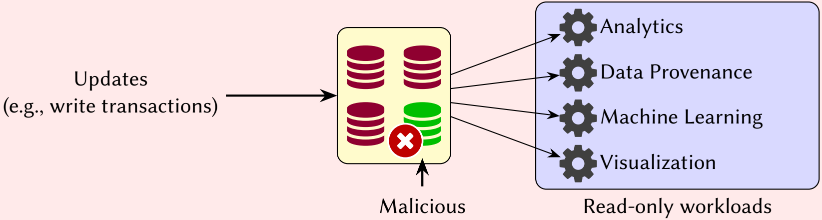
Efficient cluster-sending is possible.

Ongoing work: Initial results

- Paper: DISC 2019 (doi:10.4230/LIPIcs.DISC.2019.45).
- Technical Report: `https://arxiv.org/abs/1908.01455`.

The Byzantine learner problem

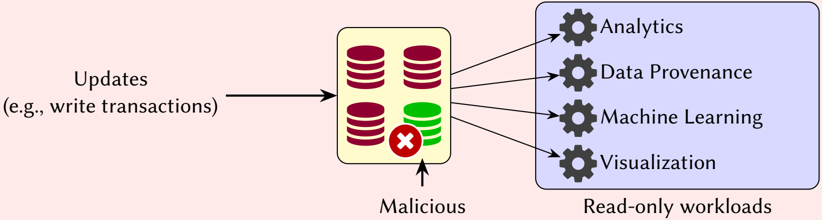
Vision: Specializing for read-only workloads



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 *information dispersal algorithm* can encode v in $\mathbf{n}$ pieces v' such that v can be *decoded* from every set of $\mathbf{n} - \mathbf{f}$ such pieces.

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

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

Theorem (Rabin)

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.

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.

The delayed-replication algorithm

Idea: C sends a Blockchain to learner L

1. Partition the Blockchain in sequences S of $\mathbf{n}$ 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.

The delayed-replication algorithm

Idea: C sends a Blockchain to learner L

1. Partition the Blockchain in sequences S of $\mathbf{n}$ 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 $\mathbf{n} - \mathbf{f}$ distinct pieces and decodes S.*

The delayed-replication algorithm

Idea: C sends a Blockchain to learner L

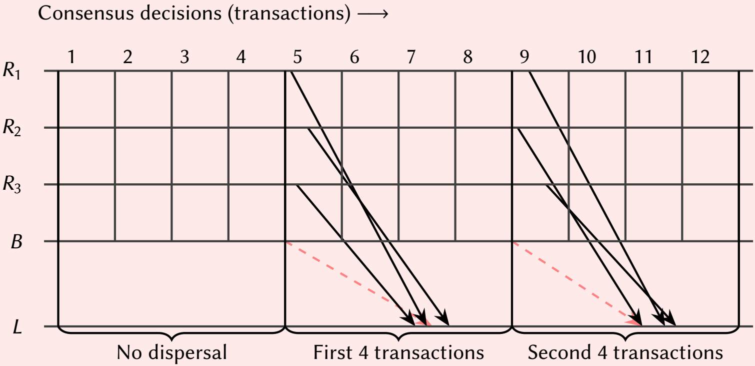
1. Partition the Blockchain in sequences S of $\mathbf{n}$ transactions.
2. Replica $R_i \in C$ encodes S into the i-th IDA piece S_i.
3. Replica $R_i \in C$ sends S_i with a checksum $C_i(S)$ of S to L.
4. *L receives at least $\mathbf{n} - \mathbf{f}$ distinct pieces and decodes S.*

Observations ($\mathbf{n} > 2\mathbf{f}$)

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

$$B \leq \mathbf{n}\left(\frac{\|S\|}{\mathbf{n} - \mathbf{f}} + 1 + c\right) < 2\|S\| + \mathbf{n} + \mathbf{n}c = \mathcal{O}\left(\|S\| + c\mathbf{n}\right).$$

Communication by the delayed-replication algorithm



Decoding S using simple checksums ($\mathbf{n} > 2\mathbf{f}$)

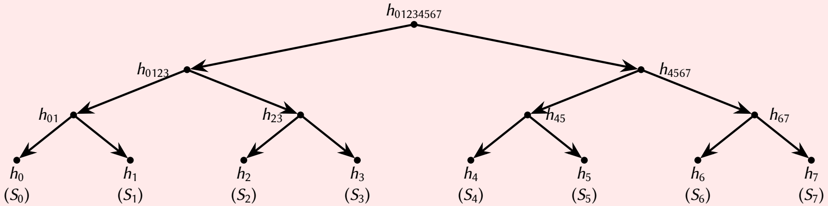
- Use checksums hash(S).
- The $\mathbf{n} - \mathbf{f}$ non-faulty replicas will provide correct *pieces*.
- At least $\mathbf{n} - \mathbf{f} > \mathbf{f}$ messages with correct *checksums*.
- Received some *forged* pieces?
  - Decoding yields S'.
  - hash(S') $\neq$ hash(S).
  - Use other pieces.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



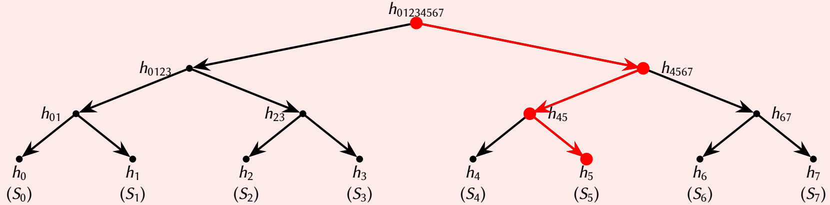
Construct a Merkle tree for pieces $S_0, \ldots, S_7$.

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).



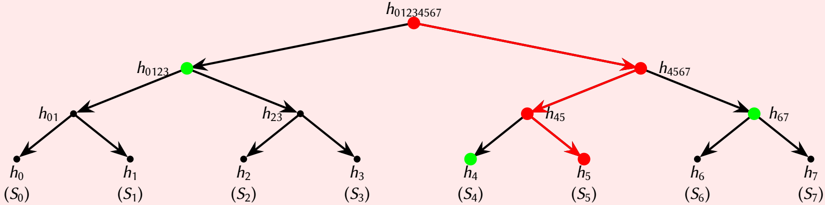
Determine the path from root to S_5.

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).



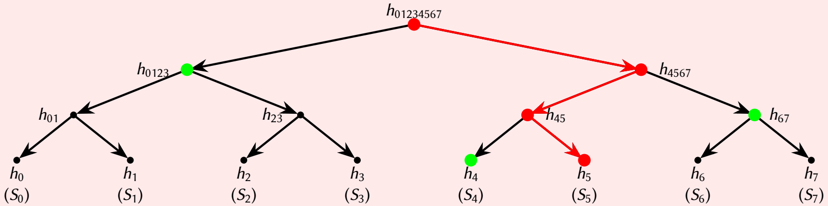
Select *root* and *neighbors*: $C_5(S) = [h_4, h_{67}, h_{0123}, h_{01234567}]$.

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 ($\mathbf{n} > 2\mathbf{f}$)

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*

$$\mathcal{O}\left(\|\mathcal{T}\|\left(\frac{\mathbf{n}}{\mathbf{n}-\mathbf{f}}\right) + |\mathcal{T}|\log \mathbf{n}\right) = \mathcal{O}\left(\|\mathcal{T}\| + |\mathcal{T}|\log \mathbf{n}\right);$$

3. *L will only need at most $|\mathcal{T}|/\mathbf{n}$ decode steps;*
4. *R will sent at most $|\mathcal{T}|/\mathbf{n}$ messages to L of size*

$$\mathcal{O}\left(\frac{\|\mathcal{T}\|}{\mathbf{n}-\mathbf{f}} + \frac{|\mathcal{T}|\log \mathbf{n}}{\mathbf{n}}\right) = \mathcal{O}\left(\frac{\|\mathcal{T}\| + |\mathcal{T}|\log \mathbf{n}}{\mathbf{n}}\right).$$

Conclusion

Efficient Byzantine learning is possible.

Blockchain applications

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

Ongoing work: Initial results

- Paper: ICDT 2020.

About us

- Jelle Hellings — https://jhellings.nl/.
- ExpoLab Creativity Unfolded — https://expolab.org/.
- ResilientDB Security, Privacy Reloaded — https://resilientdb.com/.

References I

Ittai Abraham et al. "Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience". In: *Financial Cryptography and Data Security*. Springer International Publishing, 2019, pp. 320–334. DOI: 10.1007/978-3-030-32101-7_20.

Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. "RBFT: Redundant Byzantine Fault Tolerance". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53.

Pierre-Louis Aublin et al. "The Next 700 BFT Protocols". In: *ACM Transactions on Computer Systems* 32.4 (2015), 12:1–12:45. DOI: 10.1145/2658994.

Eric Brewer. "CAP twelve years later: How the "rules" have changed". In: *Computer* 45.2 (2012), pp. 23–29. DOI: 10.1109/MC.2012.37.

Eric A. Brewer. "Towards Robust Distributed Systems (Abstract)". In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. ACM, 2000, pp. 7–7. DOI: 10.1145/343477.343502.

References II

Christian Cachin and Marko Vukolic. "Blockchain Consensus Protocols in the Wild (Keynote Talk)". In: *31st International Symposium on Distributed Computing*. Vol. 91. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 1:1–1:16. DOI: 10.4230/LIPIcs.DISC.2017.1.

Miguel Castro. "Practical Byzantine Fault Tolerance". PhD thesis. Massachusetts Institute of Technology, 2001. URL: http://hdl.handle.net/1721.1/86581.

Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. USENIX Association, 1999, pp. 173–186.

Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Transactions on Computer Systems* 20.4 (2002), pp. 398–461. DOI: 10.1145/571637.571640.

Richard A. DeMillo, Nancy A. Lynch, and Michael J. Merritt. "Cryptographic Protocols". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. ACM, 1982, pp. 383–400. DOI: 10.1145/800070.802214.

References III

Tien Tuan Anh Dinh et al. "Untangling Blockchain: A Data Processing View of Blockchain Systems". In: *IEEE Transactions on Knowledge and Data Engineering* 30.7 (2018), pp. 1366–1385. DOI: 10.1109/TKDE.2017.2781227.

D. Dolev. "Unanimity in an unknown and unreliable environment". In: *22nd Annual Symposium on Foundations of Computer Science*. IEEE, 1981, pp. 159–168. DOI: 10.1109/SFCS.1981.53.

D. Dolev and H. Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM Journal on Computing* 12.4 (1983), pp. 656–666. DOI: 10.1137/0212045.

Danny Dolev. "The Byzantine generals strike again". In: *Journal of Algorithms* 3.1 (1982), pp. 14–30. DOI: 10.1016/0196-6774(82)90004-9.

Danny Dolev and Rüdiger Reischuk. "Bounds on Information Exchange for Byzantine Agreement". In: *Journal of the ACM* 32.1 (1985), pp. 191–204. DOI: 10.1145/2455.214112.

Michael J. Fischer and Nancy A. Lynch. "A lower bound for the time to assure interactive consistency". In: *Information Processing Letters* 14.4 (1982), pp. 183–186. DOI: 10.1016/0020-0190(82)90033-3.

References IV

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. In: *Journal of the ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121.

Yossi Gilad et al. “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68. DOI: 10.1145/3132747.3132757.

Seth Gilbert and Nancy Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services”. In: *SIGACT News* 33.2 (2002), pp. 51–59. DOI: 10.1145/564585.564601.

Jim Gray. “Notes on Data Base Operating Systems”. In: *Operating Systems, An Advanced Course*. Springer-Verlag, 1978, pp. 393–481. DOI: 10.1007/3-540-08755-9_9.

Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. “Brief Announcement: Revisiting Consensus Protocols through Wait-Free Parallelization”. In: *33rd International Symposium on Distributed Computing (DISC 2019)*. Vol. 146. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 44:1–44:3. DOI: 10.4230/LIPIcs.DISC.2019.44.

References V

Suyash Gupta et al. "An In-Depth Look of BFT Consensus in Blockchain: Challenges and Opportunities". In: *Proceedings of the 20th International Middleware Conference Tutorials*. ACM, 2019, pp. 6–10. DOI: 10.1145/3366625.3369437.

Jelle Hellings and Mohammad Sadoghi. "Brief Announcement: The Fault-Tolerant Cluster-Sending Problem". In: *33rd International Symposium on Distributed Computing (DISC 2019)*. Vol. 146. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 45:1–45:3. DOI: 10.4230/LIPIcs.DISC.2019.45.

Jelle Hellings and Mohammad Sadoghi. "Coordination-free Byzantine Replication with Minimal Communication Costs". In: *Proceedings of the 23th International Conference on Database Theory*. (to appear). 2020.

Maurice Herlihy. "Blockchains from a Distributed Computing Perspective". In: *Communications of the ACM* 62.2 (2019), pp. 78–85. DOI: 10.1145/3209623.

Rüdiger Kapitza et al. "CheapBFT: Resource-efficient Byzantine Fault Tolerance". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. ACM, 2012, pp. 295–308. DOI: 10.1145/2168836.2168866.

References VI

Ramakrishna Kotla et al. "Zyzzyva: Speculative Byzantine Fault Tolerance". In: *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*. ACM, 2007, pp. 45–58. DOI: 10.1145/1294261.1294267.

Ramakrishna Kotla et al. "Zyzzyva: Speculative Byzantine Fault Tolerance". In: *ACM Transactions on Computer Systems* 27.4 (2009), 7:1–7:39. DOI: 10.1145/1658357.1658358.

Leslie Lamport. "Paxos Made Simple". In: *ACM SIGACT News, Distributed Computing Column 5* 32.4 (2001), pp. 51–58. DOI: 10.1145/568425.568433.

Leslie Lamport. "The Part-time Parliament". In: *ACM Transactions on Computer Systems* 16.2 (1998), pp. 133–169. DOI: 10.1145/279227.279229.

Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176.

Jean-Philippe Martin and Lorenzo Alvisi. "Fast Byzantine Consensus". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 202–215. DOI: 10.1109/TDSC.2006.35.

References VII

Shlomo Moran and Yaron Wolfstahl. "Extended impossibility results for asynchronous complete networks". In: *Information Processing Letters* 26.3 (1987), pp. 145–151. DOI: 10.1016/0020-0190(87)90052-4.

Arvind Narayanan and Jeremy Clark. "Bitcoin's Academic Pedigree". In: *Communications of the ACM* 60.12 (2017), pp. 36–45. DOI: 10.1145/3132259.

M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults". In: *Journal of the ACM* 27.2 (1980), pp. 228–234. DOI: 10.1145/322186.322188.

Dale Skeen. *A Quorum-Based Commit Protocol*. Tech. rep. Cornell University, 1982.

Gadi Taubenfeld and Shlomo Moran. "Possibility and impossibility results in a shared memory environment". In: *Acta Informatica* 33.1 (1996), pp. 1–20. DOI: 10.1007/s002360050034.

Maofan Yin et al. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356. DOI: 10.1145/3293611.3331591.