The Stellar Consensus Protocol
A federated model for Internet-level consensus

David Mazières
Stellar Development Foundation

Wednesday, December 6, 2017
“Prof. Mazières’s contribution to this work was as a paid consultant, and was not part of his Stanford University duties or responsibilities.”
Internet payments

Say you want to send $1 from U.S. to a customer of bank$_4$ in India. Bank$_4$ may have a *nistro* account at a European bank$_3$

- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank$_3$

Some bank$_2$ may have *nistro* accounts at bank$_3$ and your bank$_1$

- Offers 0.93 EUR at bank$_3$ in exchange for 1.00 USD at bank$_1$

Goal: implement quick, secure, atomic, irreversible payment
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank₄</td>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
</tbody>
</table>

Has account at

Say you want to send $1 from U.S. to a customer of bank₄ in India
Bank₄ may have a *nostro* account at a European bank₃
- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank₃

Some bank₂ may have *nostro* accounts at bank₃ and your bank₁
- Offers 0.93 EUR at bank₃ in exchange for 1.00 USD at bank₁

Goal: implement quick, secure, atomic, irreversible payment
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank₄</td>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
<tr>
<td>bank₂</td>
<td>0.93 EUR@bank₃</td>
<td>1.00 USD@bank₁</td>
</tr>
</tbody>
</table>

Say you want to send $1 from U.S. to a customer of bank₄ in India. Bank₄ may have a *nostro* account at a European bank₃.
- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank₃.

Some bank₂ may have *nostro* accounts at bank₃ and your bank₁.
- Offers 0.93 EUR at bank₃ in exchange for 1.00 USD at bank₁.

Goal: implement quick, secure, atomic, irreversible payment.
Internet payments

<table>
<thead>
<tr>
<th>Offeror</th>
<th>Bid</th>
<th>Ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank_4</td>
<td>60.0 INR@bank_4</td>
<td>0.93 EUR@bank_3</td>
</tr>
<tr>
<td>bank_2</td>
<td>0.93 EUR@bank_3</td>
<td>1.00 USD@bank_1</td>
</tr>
</tbody>
</table>

Say you want to send $1 from U.S. to a customer of bank_4 in India. Bank_4 may have a *nosto* account at a European bank_3.

- Will disburse 60.0 INR in exchange for 0.93 EUR on deposit at bank_3.

Some bank_2 may have *nosto* accounts at bank_3 and your bank_1:

- Offers 0.93 EUR at bank_3 in exchange for 1.00 USD at bank_1.

Goal: implement quick, secure, atomic, irreversible payment.
Strawman: Two-phase commit

<table>
<thead>
<tr>
<th>Buy</th>
<th>Sell</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.93 EUR@bank₂</td>
<td>1.00 USD@bank₁</td>
</tr>
<tr>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
</tbody>
</table>

Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction
Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank₂ can abort transaction if its offer no longer valid
**Strawman: Two-phase commit**

*atomic transaction*

<table>
<thead>
<tr>
<th>Buy</th>
<th>Sell</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.93 EUR@bank₂</td>
<td>1.00 USD@bank₁</td>
</tr>
<tr>
<td>60.0 INR@bank₄</td>
<td>0.93 EUR@bank₃</td>
</tr>
</tbody>
</table>

**Decompose payment into a series of trades**
- Effectively exchanging deposits at one bank for ones at another

**Combine them into atomic transaction**

**Use well-known two-phase commit protocol across 4 banks**
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank₂ can abort transaction if its offer no longer valid
Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank₂ can abort transaction if its offer no longer valid
Strawman: Two-phase commit

Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
- E.g., bank$_2$ can abort transaction if its offer no longer valid
Strawman: Two-phase commit

Decompose payment into a series of trades
- Effectively exchanging deposits at one bank for ones at another

Combine them into atomic transaction

Use well-known two-phase commit protocol across 4 banks
- Send transaction to every institution concerned
- Commit only if all participants unanimously vote to do so
  - E.g., bank₂ can abort transaction if its offer no longer valid
The problem: Failure

What if bank$_2$ disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank$_2$ lies and changes vote?
- Bank$_2$ might convince bank$_1$ of commit and bank$_3$ of abort…
- Now bank$_2$ receives USD at bank$_1$ without paying EUR at bank$_3$!
- Note bank$_4$ can collude with bank$_2$, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank$_2$ disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank$_2$ lies and changes vote?
- Bank$_2$ might convince bank$_1$ of commit and bank$_3$ of abort…
- Now bank$_2$ receives USD at bank$_1$ without paying EUR at bank$_3$!
- Note bank$_4$ can collude with bank$_2$, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank₂ disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank₂ lies and changes vote?
- Bank₂ might convince bank₁ of commit and bank₃ of abort…
- Now bank₂ receives USD at bank₁ without paying EUR at bank₃!
- Note bank₄ can collude with bank₂, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
The problem: Failure

What if bank\textsubscript{2} disappears mid transaction?
- Don’t know whether or when it will come back online…
- Now your original dollar is tied up pending transaction resolution

What if bank\textsubscript{2} lies and changes vote?
- Bank\textsubscript{2} might convince bank\textsubscript{1} of commit and bank\textsubscript{3} of abort…
- Now bank\textsubscript{2} receives USD at bank\textsubscript{1} without paying EUR at bank\textsubscript{3}!
- Note bank\textsubscript{4} can collude with bank\textsubscript{2}, or create chain of “Sybil” banks
- Any majority-based scheme is useless with unknown market makers

We need secure transactions across unknown, untrusted parties
Is this a job for blockchain?
What blockchain really gives us

1. Coin distribution
   - Distribute new tokens or “cryptocoins” while limiting supply

2. Irreversible transactions*
   - Can securely exchange or transfer purely digital tokens

What if bank\textsubscript{1}, bank\textsubscript{4} issue digital tokens representing USD, INR?
   - Would blockchain give us irreversible fiat money transactions?

*under certain assumptions
What blockchain really gives us

1. Coin distribution
   - Distribute new tokens or “cryptocoins” while limiting supply

2. Irreversible transactions*
   - Can securely exchange or transfer purely digital tokens

What if bank\textsubscript{1}, bank\textsubscript{4} issue digital tokens representing USD, INR?
   - Would blockchain give us irreversible fiat money transactions?

*under certain assumptions
What if bank$_1$, bank$_4$ issue digital *tokens* representing USD, INR?
- Would blockchain give us irreversible fiat money transactions?

*under certain assumptions*
Bitcoin’s key insight: Mining

Definition (Mining)

Obtaining cryptocoins as a reward for making digital transactions harder to reverse.

Mutually-reinforcing solution to coin distribution+irreversibility

Proof-of-work-based mining (popularized by Bitcoin)
- Solve hard to compute but easy to verify function on transaction set
- To reverse transaction, attacker’s work is as hard as miners’
- Currently Bitcoin miners making ~$30M/day in aggregate

Proof-of-storage or -memory (burn non-computation resource)

Proof-of-stake-based mining (many variants)
Blockchain forks

In July 2016, Ethereum executed an irregular state change
- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became Ethereum Classic

More recently Bitcoin split (Bitcoin, Bitcoin cash, Bitcoin gold…)

What would this mean for token counterparties?
- Could bank₁ be liable for twice as many digital dollars as it issued?
- Spoils of defrauding bank₁ might exceed mining rewards!
Blockchain forks

In July 2016, Ethereum executed an irregular state change
- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became *Ethereum Classic*

More recently Bitcoin split (Bitcoin, Bitcoin cash, Bitcoin gold…)
What would this mean for token counterparties?
- Could bank₁ be liable for twice as many digital dollars as it issued?
- Spoils of defrauding bank₁ might exceed mining rewards!
In July 2016, Ethereum executed an irregular state change
- 85% of miners opted to bail out DAO contract (lost $50M to bug)
- Remaining miners kept original rules, became *Ethereum Classic*

More recently Bitcoin split (Bitcoin, Bitcoin cash, Bitcoin gold...)

What would this mean for token counterparties?
- Could bank₁ be liable for twice as many digital dollars as it issued?
- Spoils of defrauding bank₁ might exceed mining rewards!
Don’t need mining for digital money tokens backed by banks
Instead, wait for token counterparties to commit transactions
- They commit only when their counterparties do, and so forth
Guarantees you agree with everyone you depend on
- E.g., never diverge from bank₁, bank₄ where you redeem USD and INR
- Through transitivity anyone you’d ever care about will agree
Don’t need mining for digital money tokens backed by banks
Instead, wait for token counterparties to commit transactions
- They commit only when their counterparties do, and so forth

Guarantees you agree with everyone you depend on
- E.g., never diverge from bank\(_1\), bank\(_4\) where you redeem USD and INR
- Through transitivity anyone you’d ever care about will agree
Insight: the Internet hypothesis

The Internet is a globally connected network
- Structure results from individual peering & transit relationships
- Decentralized decisions could have yielded many internets, but didn’t
- Transitively, everyone wants to talk to everyone

Hypothesis: counterparty relationships transitively converge
- Clearing houses are effectively the tier one ISPs of banking
Outline

Consensus background

Voting and neutralization

Federated Byzantine agreement (FBA)

The Stellar consensus protocol (SCP)
The consensus problem

Goal: For multiple agents to agree on an output value

Each agent starts with an input value
- In payments, value is an ordered batch of transactions to execute

Agents communicate following some consensus protocol
- Use protocol to agree on one of the agent’s input values

Once decided, agents output the chosen value
- Output is write-once (an agent cannot change its value)
Recall agents chose value 9 in last example
But a network outage might be indistinguishable from $v_2$ failing
If protocol fault tolerant, $v_1$ and $v_3$ might decide to output 7
Once network back, $v_2$ must also output 7

**Definition (Bivalent)**
An execution of a consensus protocol is in a **bivalent** state when the network can affect which value agents choose.
Univalent and stuck states

**Definition (Univalent, Valent)**
An execution of a consensus protocol is in a **univalent** state when only one output value is possible. If that value is $i$, call the state $i$-**valent**.

**Definition (Stuck)**
An execution of a [broken] consensus protocol is in a **stuck** state when one or more non-faulty nodes can never output a value.

Recall output is write once and all outputs must agree

- Hence, no output is possible in bivalent state

If an execution starts in a bivalent state and terminates, it must at some point reach a univalent state
Consider a terminating execution of a bivalent system

Let $m$ be the last message received in a bivalent state
- Since $m$ caused transition to univalent state, call it *deciding message*

Suppose the network had delayed $m$
- Other messages could cause transitions to other bivalent states
- Then, receiving $m$ might no longer lead to a univalent state
- In this case, we say $m$ has been *neutralized*

**Overview of FLP proof.**
1. There are bivalent starting configurations.
2. Fault tolerance $\Rightarrow$ network can neutralize any deciding msg
3. Hence, the system can remain bivalent in perpetuity.
Outline

Consensus background

Voting and neutralization

Federated Byzantine agreement (FBA)

The Stellar consensus protocol (SCP)
Suppose you have $N$ nodes with fail-stop behavior

Pick a quorum size $T > N/2$

If any $T$ nodes (a quorum) all vote for a value, output that value
  - E.g., Quorum A unanimously votes for 9, okay to output 9
  - Nodes cannot change their vote
  - Any two quorums intersect $\Rightarrow$ agreement

Problem: stuck states
  - Node failure could mean not everyone learns of unanimous quorum
  - Split vote could make unanimous quorum impossible
Straw man consensus: Vote on value

Suppose you have $N$ nodes with fail-stop behavior.

Pick a quorum size $T > N/2$.

If any $T$ nodes (a quorum) all vote for a value, output that value.

- E.g., Quorum $A$ unanimously votes for 9, okay to output 9.
- Nodes cannot change their vote.
- Any two quorums intersect $\Rightarrow$ agreement.

Problem: stuck states.

- Node failure could mean not everyone learns of unanimous quorum.
- Split vote could make unanimous quorum impossible.
Suppose you have $N$ nodes with fail-stop behavior

Pick a quorum size $T > N/2$

If any $T$ nodes (a quorum) all vote for a value, output that value

- E.g., Quorum A unanimously votes for 9, okay to output 9
- Nodes cannot change their vote
- Any two quorums intersect $\Rightarrow$ agreement

Problem: stuck states

- Node failure could mean not everyone learns of unanimous quorum
- Split vote could make unanimous quorum impossible
What voting gives us

You might get system-wide agreement or you might get stuck
Can’t vote directly on consensus question (payment committed)
What can we vote on without jeopardizing liveness?

1. Statements that never get stuck, and
2. Statements whose hold on consensus question can be broken if stuck
Statements we can vote on

1. Statements that never get stuck
   - Observation: stuck states arise because nodes can’t change votes
   - If no node ever votes against a statement, can’t get stuck

**Definition (irrefutable)**

An *irrefutable* statement is one that correct nodes never vote against.

2. Statements whose hold on consensus question can be broken
   - Recall fault tolerance requires neutralizing deciding messages

**Definition (neutralizable)**

A *neutralizable* statement is one that can be rendered irrelevant to the consensus question.

How to formulate useful yet neutralizable statements?

- Two techniques: Ballot-based and view-based neutralization
Ballot-based neutralization [Paxos]

A ballot is a pair \( \langle n, x \rangle \)
- \( n \) – a counter to ensure arbitrarily many ballots exist
- \( x \) – a candidate output value for the consensus protocol

Vote to commit or abort ballots (the two are contradictory)
- If a quorum votes to commit \( \langle n, x \rangle \) for any \( n \), it is safe to output \( x \)

Invariant: all committed and stuck ballots must have same \( x \)

To preserve: cannot vote to commit a ballot before preparing it
- Prepare \( \langle n, x \rangle \) by aborting all \( \langle n', x' \rangle \) with \( n' \leq n \) and \( x' \neq x \).
- Concisely encode whole set of abort votes with PREPARE message

If ballot \( \langle n, x \rangle \) stuck, neutralize by restarting with \( \langle n + 1, x \rangle \)
- Can prepare \( \langle n + 1, x \rangle \) even if \( \langle n, x \rangle \) is stuck
0. Initially, all ballots are bivalent

1. Agree that \( \langle 1, g \rangle \) is prepared and vote to commit it

2. Lose vote on \( \langle 1, g \rangle \); agree \( \langle 2, f \rangle \) prepared and vote to commit it

3. \( \langle 2, f \rangle \) is stuck, so agree \( \langle 3, f \rangle \) prepared and vote to commit it

4. See \( T \) votes to commit \( \langle 3, f \rangle \) (commit-valent) and externalize \( f \)
   - At this point nobody cares about \( \langle 2, f \rangle \)—neutralized

5. Node failure makes \( \langle 3, f \rangle \) stuck, prepare and commit \( \langle 4, f \rangle \)
Paxos example

0. Initially, all ballots are bivalent

1. Agree that \( \langle 1, g \rangle \) is prepared and vote to commit it

2. Lose vote on \( \langle 1, g \rangle \); agree \( \langle 2, f \rangle \) prepared and vote to commit it

3. \( \langle 2, f \rangle \) is stuck, so agree \( \langle 3, f \rangle \) prepared and vote to commit it

4. See \( T \) votes to commit \( \langle 3, f \rangle \) (commit-valent) and externalize \( f \)
   - At this point nobody cares about \( \langle 2, f \rangle \) — neutralized

5. Node failure makes \( \langle 3, f \rangle \) stuck, prepare and commit \( \langle 4, f \rangle \)
0. Initially, all ballots are bivalent
1. Agree that \(\langle 1, g \rangle\) is prepared and vote to commit it
2. Lose vote on \(\langle 1, g \rangle\); agree \(\langle 2, f \rangle\) prepared and vote to commit it
3. \(\langle 2, f \rangle\) is stuck, so agree \(\langle 3, f \rangle\) prepared and vote to commit it
4. See \(T\) votes to commit \(\langle 3, f \rangle\) (commit-valent) and externalize \(f\)
   - At this point nobody cares about \(\langle 2, f \rangle\) — neutralized
5. Node failure makes \(\langle 3, f \rangle\) stuck, prepare and commit \(\langle 4, f \rangle\)
**Paxos example**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**0. Initially, all ballots are bivalent**

**1. Agree that \( \langle 1, g \rangle \) is prepared and vote to commit it**

**2. Lose vote on \( \langle 1, g \rangle \); agree \( \langle 2, f \rangle \) prepared and vote to commit it**

**3. \( \langle 2, f \rangle \) is stuck, so agree \( \langle 3, f \rangle \) prepared and vote to commit it**

**4. See \( T \) votes to commit \( \langle 3, f \rangle \) (commit-valent) and externalize \( f \)**
   - At this point nobody cares about \( \langle 2, f \rangle \)—neutralized

**5. Node failure makes \( \langle 3, f \rangle \) stuck, prepare and commit \( \langle 4, f \rangle \)**
Paxos example

0. Initially, all ballots are bivalent

1. Agree that \(\langle 1, g \rangle\) is prepared and vote to commit it

2. Lose vote on \(\langle 1, g \rangle\); agree \(\langle 2, f \rangle\) prepared and vote to commit it

3. \(\langle 2, f \rangle\) is stuck, so agree \(\langle 3, f \rangle\) prepared and vote to commit it

4. See \(T\) votes to commit \(\langle 3, f \rangle\) (commit-valent) and externalize \(f\)
   - At this point nobody cares about \(\langle 2, f \rangle\)—neutralized

5. Node failure makes \(\langle 3, f \rangle\) stuck, prepare and commit \(\langle 4, f \rangle\)
Paxos example

0. Initially, all ballots are bivalent

1. Agree that \( \langle 1, g \rangle \) is prepared and vote to commit it

2. Lose vote on \( \langle 1, g \rangle \); agree \( \langle 2, f \rangle \) prepared and vote to commit it

3. \( \langle 2, f \rangle \) is stuck, so agree \( \langle 3, f \rangle \) prepared and vote to commit it

4. See \( T \) votes to commit \( \langle 3, f \rangle \) (commit-valent) and externalize \( f \)
   - At this point nobody cares about \( \langle 2, f \rangle \) — neutralized

5. Node failure makes \( \langle 3, f \rangle \) stuck, prepare and commit \( \langle 4, f \rangle \)
View-based neutralization [Oki]

Instead of voting on op_1, \ldots directly, vote on ⟨view 1, op_1⟩, \ldots
- Each ⟨view, op⟩ selected by a single leader for view, so irrefutable
- E.g., chose leader by round-robin using view# mod N

What if votes on op_4 and op_5 are stuck (e.g., leader fails)?
- Neutralize by agreeing view 1 had only 3 meaningful operations
  - Vote to form view 2 that immediately follows ⟨view 1, op_3⟩

Failed to form view 2 (e.g., because a node wants ⟨view 1, op_4⟩)?
- Just go on to form view 3 after ⟨view 1, op_4⟩
View-based neutralization [Oki]

Instead of voting on $\text{op}_1, \ldots$ directly, vote on $\langle \text{view 1, op}_1 \rangle, \ldots$
- Each $\langle \text{view, op} \rangle$ selected by a single leader for view, so irrefutable
- E.g., chose leader by round-robin using $\text{view} \# \mod N$

What if votes on $\text{op}_4$ and $\text{op}_5$ are stuck (e.g., leader fails)?
- Neutralize by agreeing view 1 had only 3 meaningful operations
- Vote to form view 2 that immediately follows $\langle \text{view 1, op}_3 \rangle$

Failed to form view 2 (e.g., because a node wants $\langle \text{view 1, op}_4 \rangle$)?
- Just go on to form view 3 after $\langle \text{view 1, op}_4 \rangle$
View-based neutralization [Oki]

Instead of voting on $op_1, \ldots$ directly, vote on $\langle \text{view 1, } op_1 \rangle, \ldots$

- Each $\langle \text{view, op} \rangle$ selected by a single leader for view, so irrefutable
- E.g., chose leader by round-robin using $\text{view\#} \mod N$

What if votes on $op_4$ and $op_5$ are stuck (e.g., leader fails)?

- Neutralize by agreeing view 1 had only 3 meaningful operations
- Vote to form view 2 that immediately follows $\langle \text{view 1, op}_3 \rangle$

Failed to form view 2 (e.g., because a node wants $\langle \text{view 1, op}_4 \rangle$)?

- Just go on to form view 3 after $\langle \text{view 1, op}_4 \rangle$
Byzantine agreement

What if nodes may experience Byzantine failure?
- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
- Now, any two quorums to share a non-faulty node

**Safety requires:** \(# \text{ failures} \leq f_S = 2T - N - 1\)

**Liveness requires:** \(# \text{ failures} \leq f_L = N - T\)
- At least one entirely non-faulty quorum exists

Longstanding practical protocols exist [Castro’99]
- Typically \(N = 3f + 1\) and \(T = 2f + 1\) to tolerate \(f_S = f_L = f\) failures
What if nodes may experience Byzantine failure?
- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
- Now, any two quorums to share a non-faulty node

Safety requires: # failures \( \leq f_S = 2T - N - 1 \)

Liveness requires: # failures \( \leq f_L = N - T \)
- At least one entirely non-faulty quorum exists

Longstanding practical protocols exist [Castro’99]
- Typically \( N = 3f + 1 \) and \( T = 2f + 1 \) to tolerate \( f_S = f_L = f \) failures
What if nodes may experience Byzantine failure?
- Byzantine nodes can illegally change their votes
- In fail-stop case, safety required any two quorums to share a node
  - Now, any two quorums to share a non-faulty node

**Safety requires:** \# failures \( \leq f_S = 2T - N - 1 \)

**Liveness requires:** \# failures \( \leq f_L = N - T \)
  - At least one entirely non-faulty quorum exists

**Longstanding practical protocols exist [Castro’99]**
  - Typically \( N = 3f + 1 \) and \( T = 2f + 1 \) to tolerate \( f_S = f_L = f \) failures
When has a vote succeeded?

If \( f_S + 1 = 2T - N \) nodes malicious, system loses safety

Suppose \( f_S + 1 \) nodes all claim to have seen \( T \) votes for \( a \)
- Can assume system is \( a \)-valent with no loss of safety

Now say \( f_L + f_S + 1 = T \) nodes all make same assertion
- If \( > f_L \) fail, system loses liveness (0 correct nodes in whole system)
- If \( \leq f_L \) fail, \( \geq f_S + 1 \) remain able to convince rest that system \( a \)-valent
- All correct nodes believe system \( a \)-valent \( \implies \) none stuck \( \implies \) \( a \) agreed
When has a vote succeeded?

If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

Suppose $f_S + 1$ nodes all claim to have seen $T$ votes for $a$

- Can assume system is $a$-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system $a$-valent
- All correct nodes believe system $a$-valent $\implies$ none stuck $\implies a$ agreed
When has a vote succeeded?

**Quorum A**

- $V_0$
- $\ldots$
- $\text{EVIL} \text{ EVIL} \text{ EVIL}$
- $\ldots$
- $V_{N-1}$

**Quorum B**

If $f_S + 1 = 2T - N$ nodes malicious, system loses safety

Suppose $f_S + 1$ nodes all claim to have seen $T$ votes for $a$

- Can assume system is $a$-valent with no loss of safety

Now say $f_L + f_S + 1 = T$ nodes all make same assertion

- If $> f_L$ fail, system loses liveness (0 correct nodes in whole system)
- If $\leq f_L$ fail, $\geq f_S + 1$ remain able to convince rest that system $a$-valent
- All correct nodes believe system $a$-valent $\implies$ none stuck $\implies a$ agreed
When has a vote succeeded?

If \( f_S + 1 = 2T - N \) nodes malicious, system loses safety

Suppose \( f_S + 1 \) nodes all claim to have seen \( T \) votes for \( a \)
- Can assume system is \( a \)-valent with no loss of safety

Now say \( f_L + f_S + 1 = T \) nodes all make same assertion
- If \( > f_L \) fail, system loses liveness (0 correct nodes in whole system)
- If \( \leq f_L \) fail, \( \geq f_S + 1 \) remain able to convince rest that system \( a \)-valent
- All correct nodes believe system \( a \)-valent \( \implies \) none stuck \( \implies \) \( a \) agreed

We saw a quorum vote for \( a \)
When has a vote succeeded?

If \( f_S + 1 = 2T - N \) nodes malicious, system loses safety

Suppose \( f_S + 1 \) nodes all claim to have seen \( T \) votes for a

- Can assume system is \( a \)-valent with no loss of safety

Now say \( f_L + f_S + 1 = T \) nodes all make same assertion

- If \( > f_L \) fail, system loses liveness (0 correct nodes in whole system)
- If \( \leq f_L \) fail, \( \geq f_S + 1 \) remain able to convince rest that system \( a \)-valent
- All correct nodes believe system \( a \)-valent \( \Rightarrow \) none stuck \( \Rightarrow \) a agreed
Outline

Consensus background

Voting and neutralization

Federated Byzantine agreement (FBA)

The Stellar consensus protocol (SCP)
Federated Byzantine Agreement (FBA)

Majority-based voting doesn’t work against Sybil attacks
- Attacker creates fake banks just to undermine consensus

Idea: generalize Byzantine agreement so participants determine quorums in decentralized way based on their own trust

Each node $v$ picks one or more quorum slices
- $v$ only trusts quorums that are a superset of one of its slices
- E.g., include bank$_1$, bank$_4$ in all slices if you care about their tokens

Definition (Federated Byzantine Agreement System)
An FBAS is of a set of nodes $V$ and a quorum function $Q$, where $Q(v)$ is the set slices chosen by node $v$.

Definition (Quorum)
A quorum $U \subseteq V$ is a set of nodes that contains at least one slice of each of its members: $\forall v \in U, \exists q \in Q(v)$ such that $q \subseteq U$
Federated Byzantine Agreement

Majority-based voting doesn’t work against Sybil attacks
- Attacker creates fake banks just to undermine consensus

Idea: generalize Byzantine agreement so participants determine quorums in decentralized way based on their own trust

Each node $v$ picks one or more quorum slices
- $v$ only trusts quorums that are a superset of one of its slices
- E.g., include bank$_1$, bank$_4$ in all slices if you care about their tokens

**Definition (Federated Byzantine Agreement System)**

An FBAS is of a set of nodes $V$ and a quorum function $Q$, where $Q(v)$ is the set slices chosen by node $v$.

**Definition (Quorum)**

A quorum $U \subseteq V$ is a set of nodes that contains at least one slice of each of its members: $\forall v \in U, \exists q \in Q(v)$ such that $q \subseteq U$
Definition (Quorum)

A quorum $U \subseteq V$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in Q(v)$ such that $q \subseteq U$

$v_1, v_2, v_3, v_4$ is a quorum—contains a slice of each member
$v_1, v_2, v_3$ is a slice for $v_1$, but not a quorum
  - Doesn’t contain a slice for $v_2, v_3$, who demand $v_4$’s agreement
$v_1, \ldots, v_4$ is the smallest quorum containing $v_1$

$Q(v_1) = \{\{v_1, v_2, v_3\}\}$
$Q(v_2) = Q(v_3) = Q(v_4) = \{\{v_2, v_3, v_4\}\}$
Definition (Quorum)

A quorum \( U \subseteq V \) is a set of nodes that encompasses at least one slice of each of its members: \( \forall v \in U, \exists q \in Q(v) \) such that \( q \subseteq U \)

\[
\begin{align*}
Q(v_1) &= \{\{v_1, v_2, v_3\}\} \\
Q(v_2) &= Q(v_3) = Q(v_4) = \{\{v_2, v_3, v_4\}\}
\end{align*}
\]

Visualize quorum slice dependencies with arrows

\( v_2, v_3, v_4 \) is a quorum—contains a slice of each member

\( v_1, v_2, v_3 \) is a slice for \( v_1 \), but not a quorum
  - Doesn’t contain a slice for \( v_2, v_3 \), who demand \( v_4 \)’s agreement

\( v_1, \ldots, v_4 \) is the smallest quorum containing \( v_1 \)
Definition (Quorum)
A quorum $U \subseteq V$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in Q(v)$ such that $q \subseteq U$

$Q(v_1) = \{v_1, v_2, v_3\}$
$Q(v_2) = Q(v_3) = Q(v_4) = \{v_2, v_3, v_4\}$

Quorum slice for $v_1$, but not a quorum

Visualize quorum slice dependencies with arrows
$v_2, v_3, v_4$ is a quorum—contains a slice of each member
$v_1, v_2, v_3$ is a slice for $v_1$, but not a quorum
- Doesn’t contain a slice for $v_2, v_3$, who demand $v_4$’s agreement
$v_1, \ldots, v_4$ is the smallest quorum containing $v_1$
Definition (Quorum)

A quorum $U \subseteq V$ is a set of nodes that encompasses at least one slice of each of its members: $\forall v \in U, \exists q \in Q(v)$ such that $q \subseteq U$

Visualize quorum slice dependencies with arrows

$Q(v_1) = \{\{v_1, v_2, v_3\}\}$
$Q(v_2) = Q(v_3) = Q(v_4) = \{\{v_2, v_3, v_4\}\}$

$v_2, v_3, v_4$ is a quorum—contains a slice of each member
$v_1, v_2, v_3$ is a slice for $v_1$, but not a quorum
  - Doesn’t contain a slice for $v_2, v_3$, who demand $v_4$’s agreement
$v_1, \ldots, v_4$ is the smallest quorum containing $v_1$
Tiered quorum slice example

Top tier: slice is three out of \{v_1, v_2, v_3, v_4\} (including self)

Middle tier: slice is self + any two top tier nodes

Leaf tier: slice is self + any two middle tier nodes

Like the Internet, no central authority appoints top tier
- But market can decide on \textit{de facto} tier one organizations
- Don’t even require exact agreement on who is a top tier node
Tiered quorum slice example

Like the Internet, no central authority appoints top tier
- But market can decide on *de facto* tier one organizations
- Don’t even require exact agreement on who is a top tier node
Like the Internet, no central authority appoints top tier
- But market can decide on *de facto* tier one organizations
- Don’t even require exact agreement on who is a top tier node
Tiered quorum slice example

Example: Citibank pays $1,000,000,000 to $v_7$
- Colludes to reverse transaction and double-spend same money to $v_8$
- Stellar & EFF won’t revert, so ACLU cannot accept and $v_8$ won’t either
Example: Citibank pays $1,000,000,000 to $7
- Colludes to reverse transaction and double-spend same money to $8
- Stellar & EFF won’t revert, so ACLU cannot accept and $8 won’t either
I don’t believe anything unless EFF or Stellar does

Example: Citibank pays $1,000,000,000 to \( v_7 \)
- Colludes to reverse transaction and double-spend same money to \( v_8 \)
- Stellar & EFF won’t revert, so ACLU cannot accept and \( v_8 \) won’t either
FBAS failure is per-node

Each node is either **well-behaved** or **ill-behaved**

All ill-behaved nodes have **failed**

Enough ill-behaved nodes can cause well-behaved nodes to fail
  - Bad: well-behaved nodes blocked from any progress (safe but not live)
  - Worse: well-behaved nodes in divergent states (not safe)

Well-behaved nodes are **correct** if they have not failed
Optimal FBA fault tolerance

For safety, every two quorums must share a correct node
- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can’t guarantee safety
- Call necessary property *quorum intersection despite ill-behaved nodes*

For liveness, correct nodes must form a quorum
- Otherwise, depend on failed nodes to reach agreement
Optimal FBA fault tolerance

For safety, every two quorums must share a correct node
- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can’t guarantee safety
- Call necessary property \textit{quorum intersection despite ill-behaved nodes}

For liveness, correct nodes must form a quorum
- Otherwise, depend on failed nodes to reach agreement
For safety, every two quorums must share a correct node
- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can’t guarantee safety
- Call necessary property \textit{quorum intersection despite ill-behaved nodes}

For liveness, correct nodes must form a quorum
- Otherwise, depend on failed nodes to reach agreement
Optimal FBA fault tolerance

For safety, every two quorums must share a correct node
- Conceptually remove all ill-behaved nodes from all slices
- If two disjoint quorums result, can’t guarantee safety
- Call necessary property quorum intersection despite ill-behaved nodes

For liveness, correct nodes must form a quorum
- Otherwise, depend on failed nodes to reach agreement
Outline

Consensus background

Voting and neutralization

Federated Byzantine agreement (FBA)

The Stellar consensus protocol (SCP)
First general FBA protocol
Guarantees safety when you have quorum intersection despite ill-behaved nodes (qidin)
  - This is optimal—otherwise no FBA protocol can guarantee safety
  - I.e., you may regret your choice of quorum slices, but you won’t regret choosing SCP over other Byzantine agreement protocols

Guarantees a well-behaved quorum will not get stuck

Core idea: *federated voting*
  - Nodes exchanges vote messages to agree on statements
  - Every vote specifies quorum slices (implicit in some diagrams)
  - Allows dynamic quorum discovery while assembling votes
The Stellar Consensus Protocol [SCP]

First general FBA protocol
Guarantees safety when you have quorum intersection despite ill-behaved nodes (qidin)
- This is optimal—otherwise no FBA protocol can guarantee safety
- I.e., you may regret your choice of quorum slices, but you won’t regret choosing SCP over other Byzantine agreement protocols

Guarantees a well-behaved quorum will not get stuck
Core idea: *federated voting*
- Nodes exchanges vote messages to agree on statements
- Every vote specifies quorum slices (implicit in some diagrams)
- Allows dynamic quorum discovery while assembling votes
Ratifying statements

\[ \text{vote } a \quad \text{vote } a \quad \text{vote } a \]

\begin{align*}
\text{v}_1 & & \text{v}_2 & & \text{v}_3 & & \text{Quorum} \\
\end{align*}

**Definition (ratify)**

A quorum \( U \) ratifies a statement \( a \) iff every member of \( U \) votes for \( a \). A node \( v \) ratifies \( a \) iff \( v \) is a member of a quorum \( U \) that ratifies \( a \).

Well-behaved nodes cannot vote for contradictory statements

Theorem: w. qidin, won’t ratify contradictory statements

Problem: even in a well-behaved quorum, some node \( v \) may be unable to ratify some statement \( a \) after other nodes do

- \( v \) or nodes in \( v \)'s slices might have voted against \( a \), or
- Some nodes that voted for \( a \) may subsequently have failed
Federated voting has same possible outcomes as regular voting

Apply the same reasoning as in centralized voting?
- Premise was whole system couldn’t fail; now failure is per node
- Cannot assume correctness of quorums you don’t belong to

First-hand ratification now the only way to know system \(a\)-valent
- How to agree on statement \(a\) even after voting against it?
- How to know system has agreed on \(a\)?
Federated voting has same possible outcomes as regular voting

Apply the same reasoning as in centralized voting?
- Premise was whole system couldn’t fail; now failure is per node
- Cannot assume correctness of quorums you don’t belong to

First-hand ratification now the only way to know system $a$-valent
- How to agree on statement $a$ even after voting against it?
- How to know system has agreed on $a$?

We saw a quorum vote for $a$
Federated voting has same possible outcomes as regular voting

Apply the same reasoning as in centralized voting? No!
- Premise was whole system couldn’t fail; now failure is per node
- Cannot assume correctness of quorums you don’t belong to

First-hand ratification now the only way to know system \( a \)-valent
- How to agree on statement \( a \) even after voting against it?
- How to know system has agreed on \( a \)?
Accepting statements

**System is \(a\)-valent**

\[ Q(v_1) = \{ \{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\} \} \]

**What if one node in each of \(v_1\)'s slices says system is \(a\)-valent?**

- Either true or \(v_1\) not member of any well-behaved quorum (no liveness)

**Definition (accept)**

Node \(v\) **accepts** a statement \(a\) consistent with history iff either:

1. A quorum containing \(v\) each either voted for or accepted \(a\), or
2. Each of \(v\)'s quorum slices has a node claiming to accept \(a\).

#2 lets a node accept a statement after voting against it, but...

1. Still no guarantee all supposedly live nodes can accept a statement
2. Can accept diverging statements even with qidin
   (“intersects all slices” \(\approx f_L + 1\) centralized nodes when we want \(f_S + 1\)
Accepting statements

What if one node in each of $v_1$’s slices says system is $α$-valent?
- Either true or $v_1$ not member of any well-behaved quorum (no liveness)

Definition (accept)
Node $v$ accepts a statement $a$ consistent with history iff either:
1. A quorum containing $v$ each either voted for or accepted $a$, or
2. Each of $v$’s quorum slices has a node claiming to accept $a$.

#2 lets a node accept a statement after voting against it, but…
1. Still no guarantee all supposedly live nodes can accept a statement
2. Can accept diverging statements even with qidin (“intersects all slices” $\approx f_L + 1$ centralized nodes when we want $f_S + 1$)
Confirmation

Idea: Hold a second vote on the fact that the first vote succeeded

Definition (confirm)
A quorum **confirms** a statement \( a \) by ratifying the statement “We accepted \( a \).” A node **confirms** \( a \) iff it is in such a quorum.

Solves problem 2 (suboptimal safety) w. straight-up ratification
Solves problem 1 (live nodes unable to accept)
- Supposedly live nodes may vote against accepted statements
- Won’t vote against the *fact* that those statements were accepted
- Hence, the fact of acceptance is irrefutable

Theorem: If 1 node in well-behaved quorum confirms \( a \), all will
A node $\nu$ that locally confirms $a$ knows system has agreed on $a$

- If qidin, well-behaved nodes can’t contradict $a$
- If $\nu$ in well-behaved quorum, whole quorum will eventually confirm $a$
Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into composite value $x$
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into *composite* value $x$
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
SCP: High-level view

Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into composite value $x$
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
SCP: High-level view

Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into composite value $x$
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into composite value $x$
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
Phase 1: Nomination (c.f. async reliable broadcast)
- Nodes nominate values until at least one value confirmed nominated
- Nomination irrefutable—can’t vote against nominating & get stuck
- Propagate values and converge on set of nominated values
- Deterministically combine nominated values into composite value \( x \)
- Complication: impossible to know when protocol converges [FLP]

Phase 2: Balloting
- Similar to Byzantine Paxos, but with federated voting
- Works (less efficiently) even when Phase 1 hasn’t converged
The Stellar Network

Implementation of SCP used by Stellar payment network
- ~20 nodes, configured to kick off consensus every ~5 seconds

Open network anyone can join
- Of course, joining doesn’t mean others will trust you

In use today for international payments
- No USD yet, but EUR, NGN, PHP, CNY, JPY, with more currencies coming soon
## Comparison to other approaches

<table>
<thead>
<tr>
<th>mechanism</th>
<th>open network</th>
<th>low latency</th>
<th>flexible trust</th>
<th>asympt. security</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Byzantine agr.</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>proof-of-work</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>proof-of-stake</td>
<td>✓</td>
<td>maybe</td>
<td></td>
<td>maybe</td>
</tr>
</tbody>
</table>

### Use traditional Byzantine agreement over closed server set?
- Paranoid users will check outside audits anyway (ersatz FBA)
- Might as well formalize the arrangement to get optimal safety

### Use (proof-of-work) blockchain for Internet-level consensus?
- Consensus intricately tied up with coin distribution & incentives
- Incentives insufficient or ill-suited to other applications (fiat currency)
Another application: timestamping

Certificate Transparency provides trusted logs alongside CAs
- Generalize CT logging to leverage logs for timestamping documents?

Problem: which log to use?
- Problem: different people trust different logs
- Don’t know in advance to whom you will need to prove timestamp

What if your log choice proves untrustworthy?
Internet-level consensus on timestamps avoids problem
Another application: timestamping

Certificate Transparency provides trusted logs alongside CAs - Generalize CT logging to leverage logs for timestamping documents?

Problem: which log to use?
- Problem: different people trust different logs
- Don’t know in advance to whom you will need to prove timestamp

What if your log choice proves untrustworthy?

Internet-level consensus on timestamps avoids problem
Application: Software transparency

In 2016, FBI ordered Apple to sign a compromised bootloader
  - Apple appears to have refused, but how can we know for sure?

Make software updates visible through software transparency
  - Devices refuse to install updates not in public log
  - Log integrity secured through ILC

E.g., Mozilla Binary transparency could benefit from Internet-level consensus
Questions?

www.stellar.org
Cyclic quorum slice example

Traditional Byzantine agreement requires $\forall (i, j), Q(v_i) = Q(v_j)$
- Means no distinction between quorums and quorum slices

**Federated Byzantine** agreement accommodates different slices
- May even have disjoint slices if you have cycles
- Shouldn’t necessarily invalidate safety guarantees