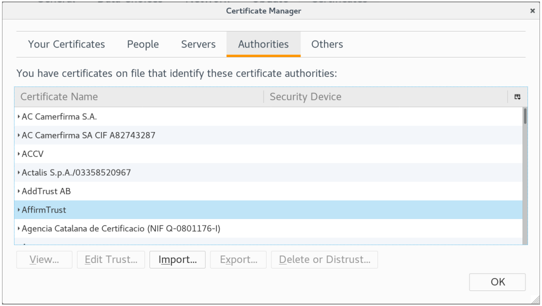
Internet-level consensus is practical

David Mazières

IETF98

Thursday, March 30, 2017

Disjunctive vs. conjunctive security



You have certificates on file that identify these certificate authorities:

- AC Camerfirma S.A.
- AC Camerfirma SA CIF A82743287
- ACCV
- Actalis S.p.A./03358520967
- AddTrust AB
- AffirmTrust
- Agencia Catalana de Certificacio (NIF Q-0801176-I)

We often require that *one* CA or *one* CT log endorse something

Today's talk: what if you want *all* CAs or *all* logs to agree?

- Who are "all" CAs or logs? E.g., 180+ Mozilla CAs w. 65+ owners?
- Different OS distributions ship different variants of root CA set
- Some organizations use in-house CAs that aren't globally trusted

This is the *Internet-level consensus* (ILC) problem

Outline

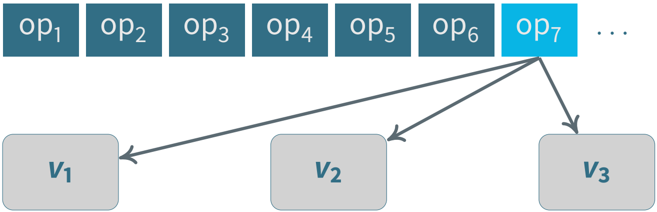
Motivation

Consensus background

Federated Byzantine Agreement (FBA)

The Stellar consensus protocol (SCP)

Consensus: The key to replication



Consensus keeps replicated data structures in sync

- All nodes agree on initial state + series of operations on state

Internet-level consensus makes history resistant to tampering

- If “whole Internet” agrees on op_7, hard to pretend it didn’t happen

Particularly powerful for replicating verifiable data structures

- Huge data collections permitting concise proofs of individual elements

Application 1: Global timestamp service

Suppose you want to obtain secure document timestamps

Idea: Generalize CT logging to leverage logs for other purposes

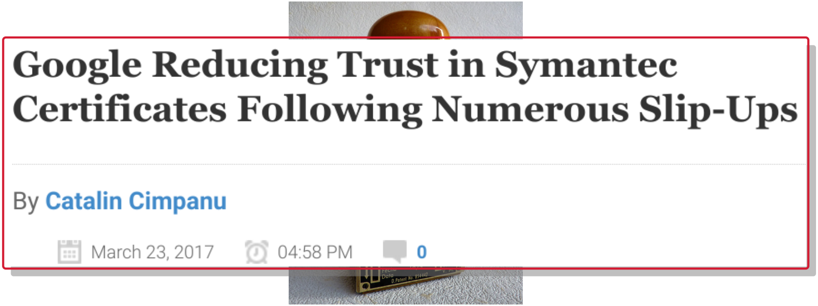
Which log to use?

- Different people will trust different logs
- Might not know in advance to whom you'll need to prove timestamp

What if your log proves untrustworthy?

Using ILC for timestamps would avoid this problem

Application 1: Global timestamp service



Google Reducing Trust in Symantec Certificates Following Numerous Slip-Ups

By Catalin Cimpanu

March 23, 2017 04:58 PM 0

Suppose you want to obtain secure document timestamps

Idea: Generalize CT logging to leverage logs for other purposes

Which log to use?

- Different people will trust different logs
- Might not know in advance to whom you'll need to prove timestamp

What if your log proves untrustworthy?

Using ILC for timestamps would avoid this problem

Application 2: Software transparency

Many package managers install digitally signed software

But really want two guarantees beyond signatures for packages:

1. You are installing the same public software as everyone else
   (not some "special" version signed by a compromised author/vendor)
2. It's not an old version with known vulnerabilities

Again, ILC can solve these problems [SPAM]

- Guarantee installed software has been publicly available for audit
- Guarantee author has not published revocation for version

Application 3: Internet payments

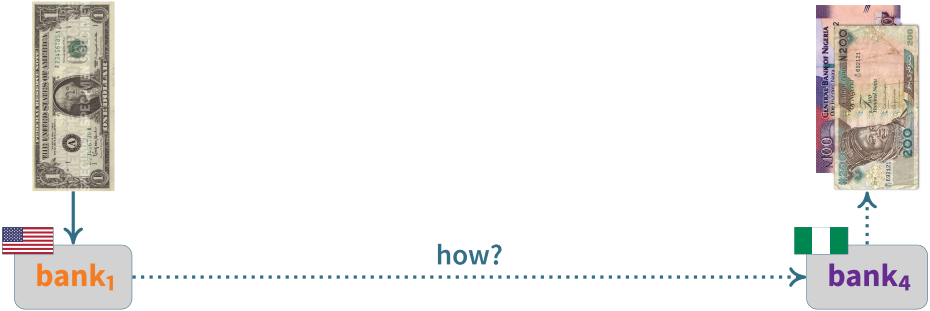
Suppose you want to send a dollar over the Internet

May require transaction across multiple financial institutions

- ILC can make such transactions secure and atomic
- Even across institutions with no prior relationship or trust

Technique in production use today by Stellar payment network

Internet payments (continued)



Say you want to send \$1 from US bank$_1$ to Nigerian bank$_4$

bank$_4$ may have a *nostro* account at some European bank$_3$

- Offers 300 NGN in exchange for 0.93 EUR on deposit at bank$_3$

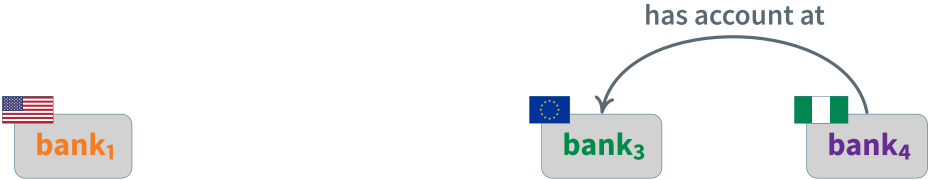
Some bank$_2$ may have *nostro* accounts at bank$_1$ and bank$_3$

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

ILC makes this whole transaction atomic and irreversible

Internet payments (continued)

Offeror	Bid	Ask
bank$_4$	300 NGN@bank$_4$	0.93 EUR@bank$_3$



Say you want to send $1 from US bank$_1$ to Nigerian bank$_4$

bank$_4$ may have a *nostro* account at some European bank$_3$

- Offers 300 NGN in exchange for 0.93 EUR on deposit at bank$_3$

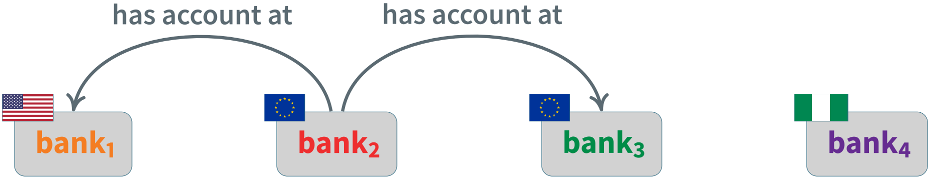
Some bank$_2$ may have *nostro* accounts at bank$_1$ and bank$_3$

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

ILC makes this whole transaction atomic and irreversible

Internet payments (continued)

Offeror	Bid	Ask
bank$_4$	300 NGN@bank$_4$	0.93 EUR@bank$_3$
bank$_2$	0.93 EUR@bank$_3$	1.00 USD@bank$_1$



Say you want to send $1 from US bank$_1$ to Nigerian bank$_4$

bank$_4$ may have a *nostro* account at some European bank$_3$

- Offers 300 NGN in exchange for 0.93 EUR on deposit at bank$_3$

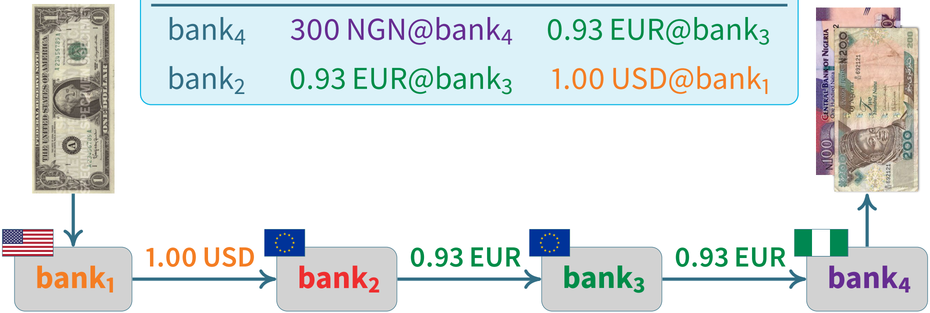
Some bank$_2$ may have *nostro* accounts at bank$_1$ and bank$_3$

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

ILC makes this whole transaction atomic and irreversible

Internet payments (continued)

Offeror	Bid	Ask
bank$_4$	300 NGN@bank$_4$	0.93 EUR@bank$_3$
bank$_2$	0.93 EUR@bank$_3$	1.00 USD@bank$_1$



Say you want to send \$1 from US bank$_1$ to Nigerian bank$_4$

bank$_4$ may have a *nostro* account at some European bank$_3$

- Offers 300 NGN in exchange for 0.93 EUR on deposit at bank$_3$

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

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

ILC makes this whole transaction atomic and irreversible

Outline

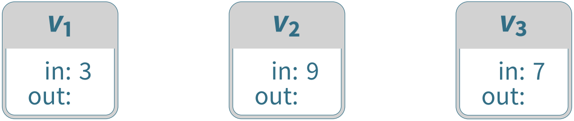
Motivation

Consensus background

Federated Byzantine Agreement (FBA)

The Stellar consensus protocol (SCP)

The consensus problem



v_1
in: 3
out:

v_2
in: 9
out:

v_3
in: 7
out:

Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. in a replicated log

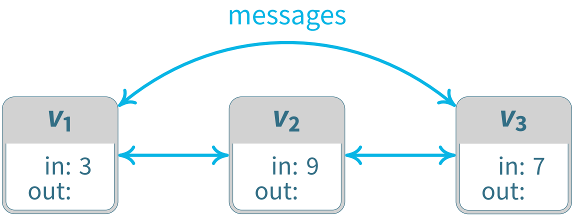
Agents communicate following some *consensus protocol*

- 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)

The consensus problem



Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. in a replicated log

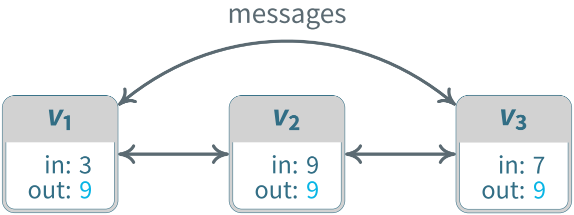
Agents communicate following some *consensus protocol*

- 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)

The consensus problem



Goal: For multiple agents to agree on an output value

Each agent starts with an input value

- Typically a candidate for the *n*th op. in a replicated log

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)

Properties of a consensus protocol

A consensus protocol provides safety iff…

- All outputs produced have the same value (*agreement*), and
- The output value equals one of the agents' inputs (*validity*)

A consensus protocol provides liveness iff…

- Eventually non-faulty agents output a value (*termination*)

A consensus protocol provides fault tolerance iff…

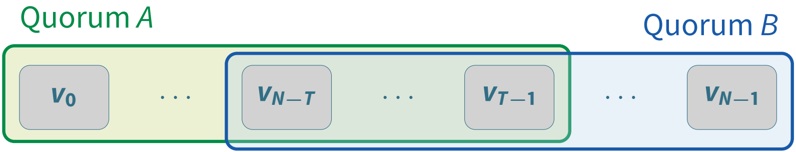
- It can recover from the failure of an agent at any point
- *Fail-stop* protocols handle agent crashes
- *Byzantine-fault-tolerant* protocols handle arbitrary agent behavior

Theorem (FLP impossibility result)

No deterministic consensus protocol can guarantee all three of safety, liveness, and fault tolerance in an asynchronous system.

Safe+fault-tolerant protocols may terminate *in practice*

Byzantine agreement



Byzantine agreement is one practical solution to consensus

- Requires participation of a *quorum* of T out of N nodes
- Faulty nodes may maliciously send contradictory messages

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

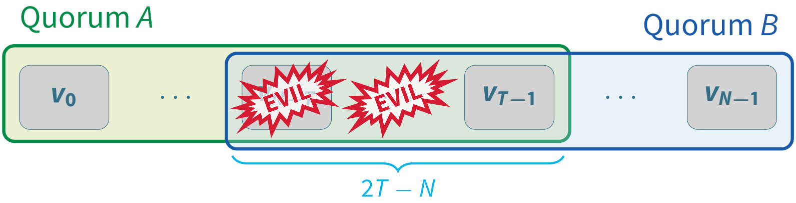
- Hence, any two quorums share a *non-faulty* node, can't lose history

***Liveness* requires at least 1 quorum: # failures $\leq f_L = N - T$**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

The problem: politically, can't enumerate the N nodes of Internet

Byzantine agreement



Byzantine agreement is one practical solution to consensus

- Requires participation of a *quorum* of T out of N nodes
- Faulty nodes may maliciously send contradictory messages

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

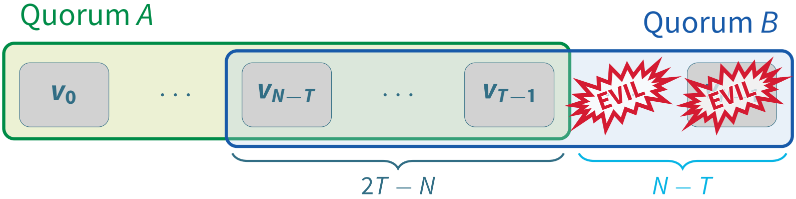
- Hence, any two quorums share a *non-faulty* node, can't lose history

Liveness **requires at least 1 quorum: # failures $\leq f_L = N - T$**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

The problem: politically, can't enumerate the N nodes of Internet

Byzantine agreement



Byzantine agreement is one practical solution to consensus

- Requires participation of a *quorum* of T out of N nodes
- Faulty nodes may maliciously send contradictory messages

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

- Hence, any two quorums share a *non-faulty* node, can't lose history

***Liveness* requires at least 1 quorum: # failures $\leq f_L = N - T$**

Typically $N = 3f + 1$ and $T = 2f + 1$ to tolerate $f_S = f_L = f$ failures

The problem: politically, can't enumerate the N nodes of Internet

Outline

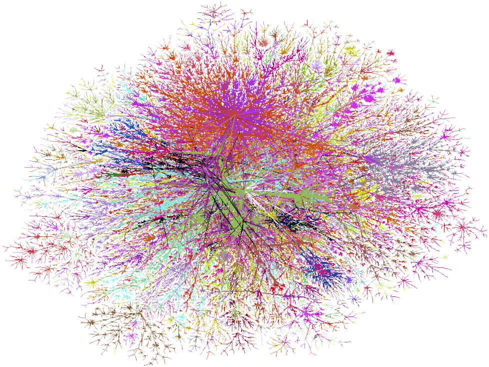
Motivation

Consensus background

Federated Byzantine Agreement (FBA)

The Stellar consensus protocol (SCP)

Byzantine agreement in an open network



How to achieve consensus without meta-consensus on *N* nodes?

Related question: how to achieve global network reachability without consensus on tier-one ISPs?

- Answer: build network out of pairwise peering & transit relationships

Idea: use pairwise trust to achieve secure global consensus

- Like inter-domain routing, though costs, branching factor will differ

Federated Byzantine Agreement (FBA)

FBA is a generalization of the Byzantine agreement problem

- Byzantine agreement without magically blessing N nodes

Participants determine quorums in decentralized way

- Each node v picks one or more *quorum slices*, where v in all its slices
- v only trusts quorums that are a superset of one of its slices

If you care about an authority, put it in all your slices

Definition (Federated Byzantine Agreement System)

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

Definition (Quorum)

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

Federated Byzantine Agreement

FBA is a generalization of the Byzantine agreement problem

- Byzantine agreement without magically blessing *N* nodes

Participants determine quorums in decentralized way

- Each node *v* picks one or more *quorum slices*, where *v* in all its slices
- *v* only trusts quorums that are a superset of one of its slices

If you care about an authority, put it in all your slices

Definition (Federated Byzantine Agreement System)

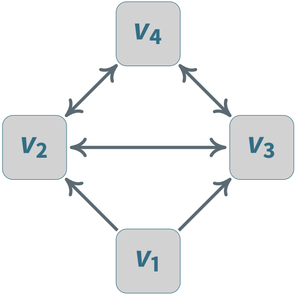
An FBAS is of a a set of nodes $\mathbf{V}$ and a quorum function $\mathbf{Q}$, where $\mathbf{Q}(v)$ is the set slices chosen by node v.

Definition (Quorum)

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

Definition (Quorum)

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



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

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

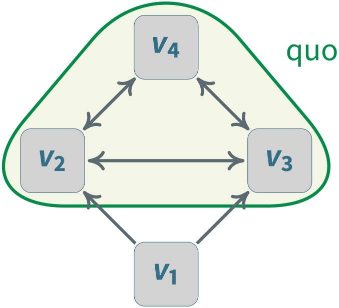
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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



quorum for v_2, v_3, v_4

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

Visualize quorum slice dependencies with arrows

v_2, v_3, v_4 **is a quorum—contains a slice of each member**

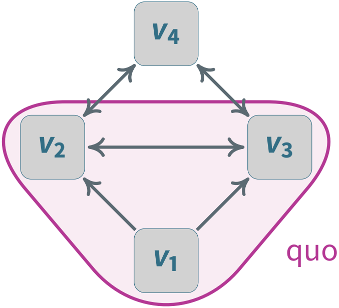
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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



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

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

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**

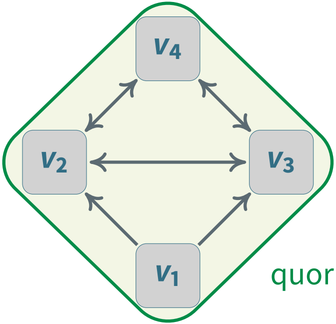
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

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



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

quorum for $v_1, \ldots, v_4$

Visualize quorum slice dependencies with arrows

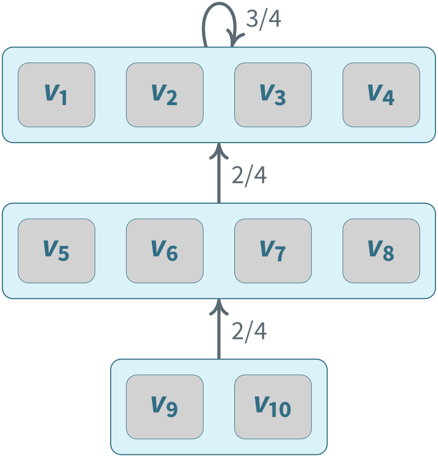
v_2, v_3, v_4 **is a quorum—contains a slice of each member**

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

Tiered quorum slice example



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

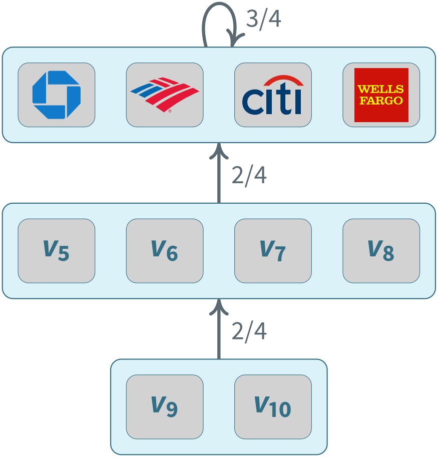
Middle tier: slice is self + any two top tier nodes

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

Like the Internet, no central authority appoints top tier

- But market can decide on *de facto* tier one organizations
- Don't even require exact agreement on who is a top tier node

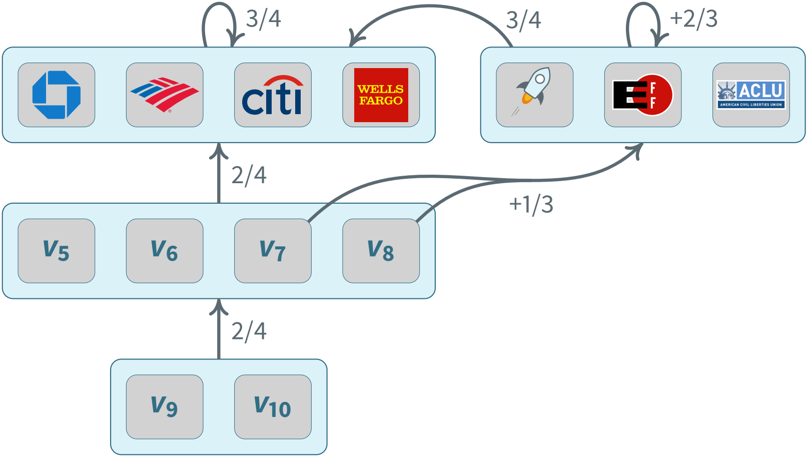
Tiered quorum slice example



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

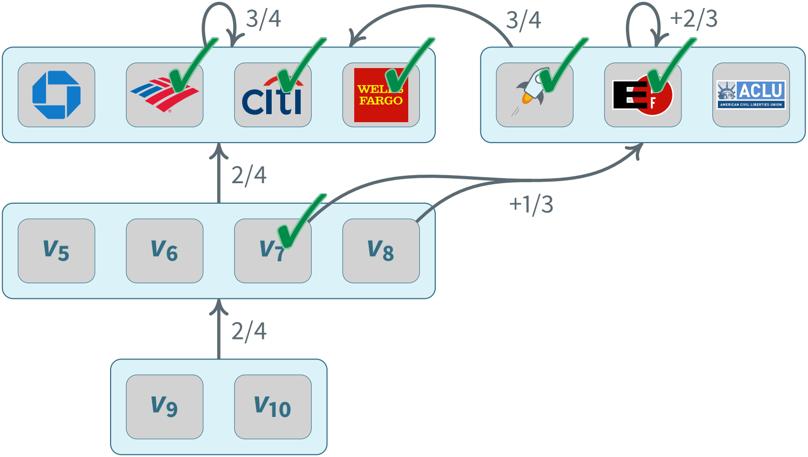
Tiered quorum slice example



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

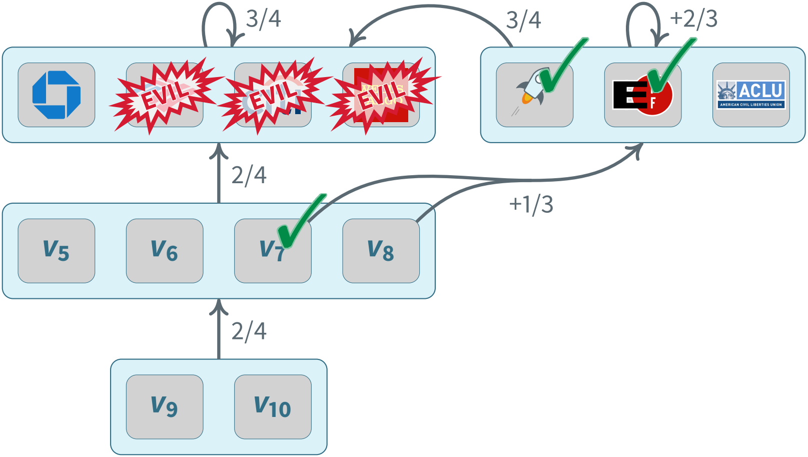
Tiered quorum slice example



Example: Citibank pays $1,000,000,000 Chase dollars 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

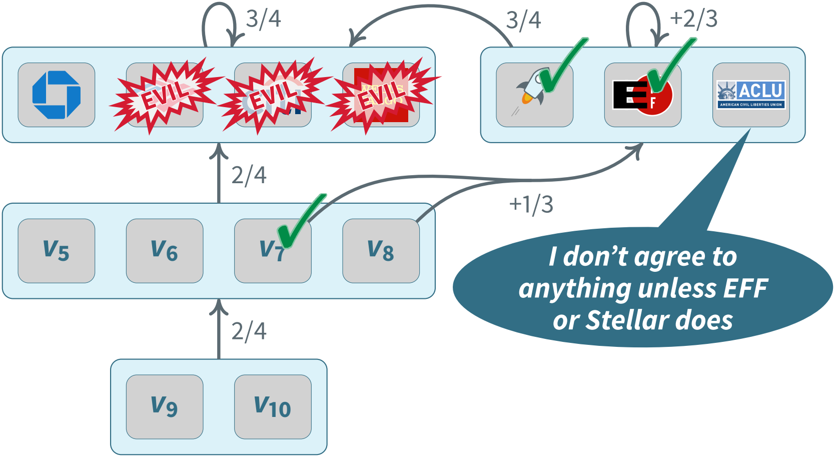
Tiered quorum slice example



Example: Citibank pays $1,000,000,000 Chase dollars 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

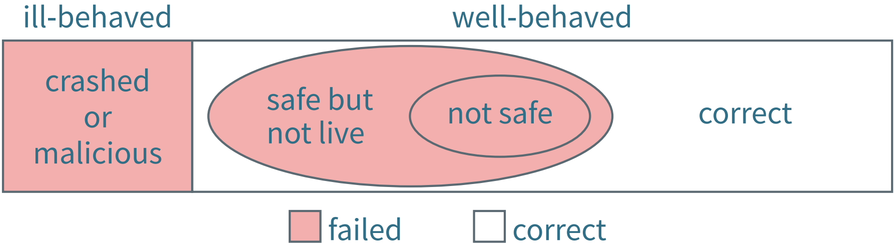
Tiered quorum slice example


3/4
EVIL
EVIL
EVIL
3/4
+2/3
ACLU
AMERICAN CIVIL LIBERTIES UNION
2/4
+1/3
v_5
v_6
v_7
v_8
I don’t agree to anything unless EFF or Stellar does
2/4
v_9
v_{10}

Example: Citibank pays $1,000,000,000 Chase dollars 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

Failure is per node in FBA



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

All ill-behaved nodes have *failed*

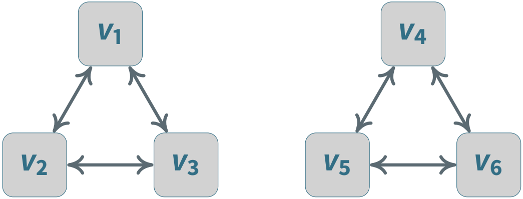
Enough ill-behaved nodes can cause well-behaved nodes to fail

- 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

What is necessary to guarantee safety?

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



$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6\}\}$

Suppose there are two entirely disjoint quorums

- Each can make progress with no communication from the other
- No way to guarantee the two externalize consistent statements

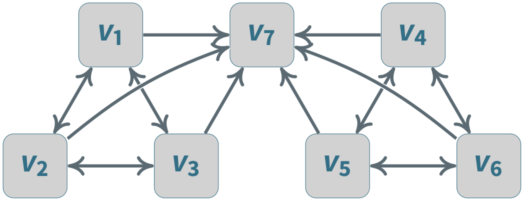
As in centralized systems, safety requires quorum intersection

Definition (Quorum intersection)

An FBAS enjoys quorum intersection when every two quorums share at least one node.

What about Byzantine failures?

$\mathbf{Q}(v_7) = \{\{v_7\}\}$



$\mathbf{Q}(v_1) =$
$\mathbf{Q}(v_2) =$
$\mathbf{Q}(v_3) =$
$\{\{v_1, v_2, v_3, v_7\}\}$

$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

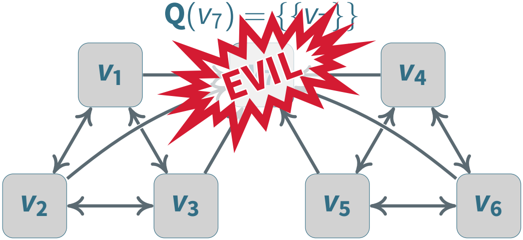
Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn’t undermine intersection
- In this example, reduces to diagram on previous slide

What about Byzantine failures?

$\mathbf{Q}(v_7) = \{\{v_7\}\}$

$\mathbf{Q}(v_1) =$
$\mathbf{Q}(v_2) =$
$\mathbf{Q}(v_3) =$
$\{\{v_1, v_2, v_3, v_7\}\}$



$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

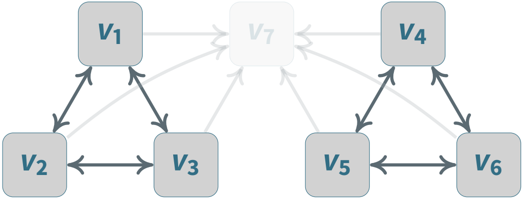
Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn't undermine intersection
- In this example, reduces to diagram on previous slide

What about Byzantine failures?

$\mathbf{Q}(v_7) = \{\{v_7\}\}$

$\mathbf{Q}(v_1) =$
$\mathbf{Q}(v_2) =$
$\mathbf{Q}(v_3) =$
$\{\{v_1, v_2, v_3, v_7\}\}$



$\mathbf{Q}(v_4) =$
$\mathbf{Q}(v_5) =$
$\mathbf{Q}(v_6) =$
$\{\{v_4, v_5, v_6, v_7\}\}$

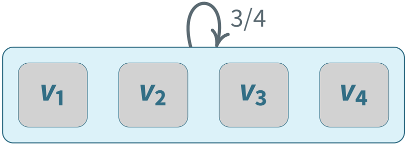
Suppose two quorums intersect only at Byzantine nodes

- Byzantine nodes behave arbitrarily
- Can feed inconsistent data to different honest nodes
- No way to guarantee safety

Necessary property for safety with Byzantine failures:
Quorum intersection *despite ill-behaved nodes*

- Means deleting ill-behaved nodes doesn't undermine intersection
- In this example, reduces to diagram on previous slide

What is necessary to guarantee liveness?



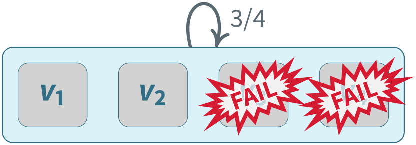
$\mathbf{Q}(v_1) = v_1$ plus two of $\{v_2, v_3, v_4\}$
$\mathbf{Q}(v_2) = v_2$ plus two of $\{v_1, v_3, v_4\}$

Suppose each of v_1's slices contains a Byzantine node

- Every quorum containing v_1 will also include a Byzantine node
- Byzantine includes crashed—might not agree to anything
- Impossible to guarantee liveness for v_1

Necessary property for liveness: Correct nodes form a quorum

What is necessary to guarantee liveness?



$\mathbf{Q}(v_1) = v_1$ plus two of $\{v_2, v_3, v_4\}$
$\mathbf{Q}(v_2) = v_2$ plus two of $\{v_1, v_3, v_4\}$

Suppose each of v_1's slices contains a Byzantine node

- Every quorum containing v_1 will also include a Byzantine node
- Byzantine includes crashed—might not agree to anything
- Impossible to guarantee liveness for v_1

Necessary property for liveness: **Correct nodes form a quorum**

Optimal failure resilience

Suppose *U* is a set of well-behaved nodes in an FBAS

- Let $\overline{U}$ be the nodes not in *U*—might be ill-behaved

An FBAS can guarantee safety for *U* only if:

1. *U* enjoys quorum intersection despite $\overline{U}$.

Can guarantee correctness (safety+liveness) for *U* only if:

1. *U* enjoys quorum intersection despite $\overline{U}$, and
2. *U* is a quorum.

Outline

Motivation

Consensus background

Federated Byzantine Agreement (FBA)

The Stellar consensus protocol (SCP)

The Stellar Consensus Protocol [SCP]

First general FBA protocol

Guarantees safety if well-behaved nodes enjoy quorum intersection despite ill-behaved nodes

- If nodes diverge, no other protocol could have guaranteed safety
- I.e., you might regret your choice of quorum slices, but you won’t regret choosing SCP over other Byzantine agreement protocols

Guarantees well-behaved quorum will not get stuck

Core idea: *federated voting*

- Nodes exchanges vote messages to agree on statements
- Every message also specifies the voter’s quorum slices
- Allows dynamic quorum discovery while assembling votes

SCP currently runs at the heart of Stellar payment network

- ~20 nodes, configured to kick off consensus every 5 seconds

SCP: High-level view

Phase 1: Nomination

- Nodes nominate values
- Nodes are guaranteed to converge on a set of nominated values
  - But don't know when, or would violate FLP
- Combine set of nominated values in deterministic way
  - E.g., union of sets of transactions & max of timestamps
- Feed combined value into balloting phase

Phase 2: Balloting

- Similar to Byzantine Paxos, but with federated voting
- Provides safety and liveness guarantees from previous slide

Comparison to other approaches

mechanism	open network	low latency	flexible trust	asympt. security
SCP	✓	✓	✓	✓
Byzantine agr.		✓	✓	✓
proof-of-work	✓			
proof-of-stake	✓	maybe		maybe

Use traditional Byzantine agreement over closed CA list for ILC?

- Those depending on outside audits will create poor-man's FBA anyway
- Might as well formalize the arrangement to get optimal safety

Use Bitcoin block chain (proof-of-work) for ILC?

- Consensus intricately tied up with coin distribution & incentives
- Incentives might be insufficient or ill-suited to CA-type applications

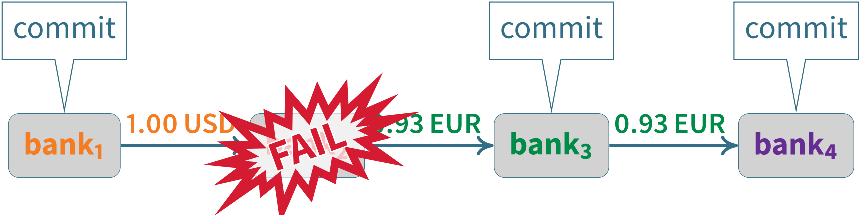
Further discussion

Questions now?

Bar BoF tonight, 7:30pm–9:00pm

Internet-level consensus mailing list:
`https://www.ietf.org/mailman/listinfo/ilc`

Without ILC, failure poses problems



What if some bank(s) disappear mid-transaction?

- Don't know whether or when missing banks will come back online…
- Other banks' funds tied up pending transaction resolution

What if bank$_2$ lies and changes vote? Or colludes with bank$_4$?

- Convince bank$_1$ of commit and bank$_3$ of abort $\Longrightarrow$ steal money

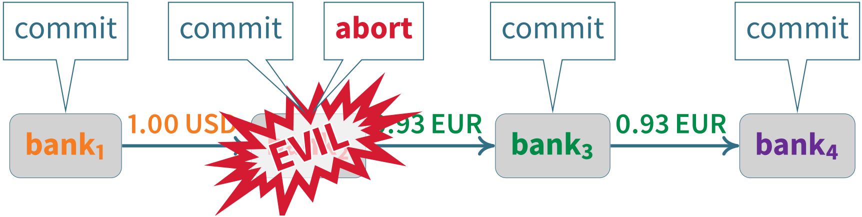
bank$_2$ shouldn't be able to cause such issues

- Other banks only know it as a customer, should limit trust

ILC leverages global set of participants to solve problem

- Even if bank$_2$ and bank$_4$ are evil, ILC can commit transaction and order it before malicious transactions cooked up by bad banks

Without ILC, failure poses problems



What if some bank(s) disappear mid-transaction?

- Don't know whether or when missing banks will come back online…
- Other banks' funds tied up pending transaction resolution

What if bank$_2$ lies and changes vote? Or colludes with bank$_4$?

- Convince bank$_1$ of commit and bank$_3$ of abort $\Longrightarrow$ steal money

bank$_2$ shouldn't be able to cause such issues

- Other banks only know it as a customer, should limit trust

ILC leverages global set of participants to solve problem

- Even if bank$_2$ and bank$_4$ are evil, ILC can commit transaction and order it before malicious transactions cooked up by bad banks