The Stellar Consensus Protocol (SCP)

draft-mazieres-dinrg-scp-04

Nicolas Barry, Giuliano Losa, David Mazières, Jed McCaleb, Stanislas Polu

IETF102

Friday, July 20, 2018

Motivation: Internet-level consensus

Atomically transact across incompatible/distrustful systems

- E.g., Transfer domain name in exchange for payment
- Can we leverage "the Internet" and its decentralized governance to create a secure, reliable two-phase commit coordinator?

Irrevocably delegate identifiers

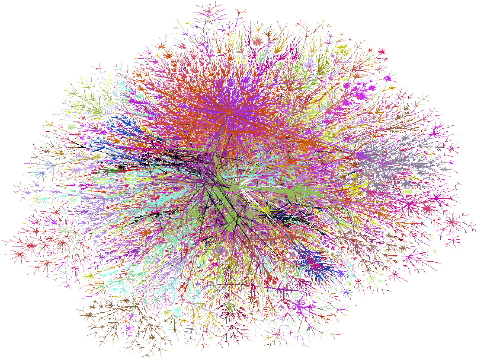
- E.g., certify email user public key w/o ability to equivocate
- Can "the Internet" enforce delegation rules?

Verify public disclosure & timestamp of information

- Build IoT device that only upgrades to public firmware
- Can "the Internet" maintain a software transparency log?

All of these can be addressed w. public append-only log

What is the Internet?



We think of IANA, ICANN, recursive delegation

- But if Google, Netflix, Amazon, Comcast, etc. moved to a parallel IP network, most people in US wouldn't care about IANA or ICANN
- People in China care about different sites—can't even reach Google

Hypothesis: all notions of the Internet transitively converge

- Inherent Brinkmanship to network build out of pairwise peering
- But huge disincentive to leaving keeps network transitively connected

Consensus based on Internet hypothesis

Idea: Everyone picks a quorum slice that speaks for the Internet

- E.g., I pick Stanford, IETF
- You pick Baidu, Wechat, Alibaba
- Alibaba and Stanford both include Google in their quorum slices
- Transitively, we both depend on Google
- Want guaranteed agreement so long as Google honest

For fault tolerance, pick multiple quorum slices

- E.g., depend on 4/5 FAANG companies
- More realistically 3/4 of servers from each of 5 FAANGs

Define quorums as transitive closure of slices

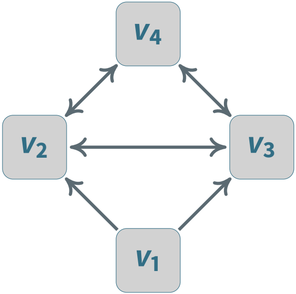
- Let $\mathbf{V}$ be all nodes, $\mathbf{Q}(v)$ be all of node v's quorum slices

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

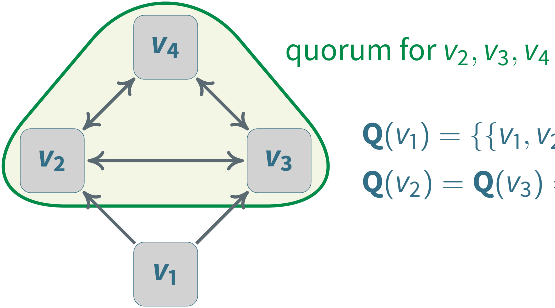
v_1, v_2, v_3 is a slice for v_1, but not a quorum

- 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 $q \subseteq U$



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

Visualize quorum slice dependencies with arrows

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

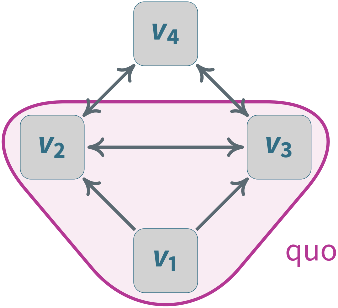
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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**

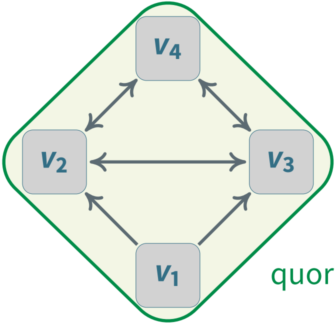
v_1, v_2, v_3 **is a slice for** v_1**, but not a quorum**

- 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

Quorum slice representation

```
struct SCPSlices {
    uint32 threshold;           // the k in k-of-n
    PublicKey validators<>;
    SCPSlices1 innerSets<>;
};
struct SCPSlices1 {
    uint32 threshold;           // the k in k-of-n
    PublicKey validators<>;
    SCPSlices2 innerSets<>;
};
struct SCPSlices2 {
    uint32 threshold;           // the k in k-of-n
    PublicKey validators<>;
};
```

Can't represent arbitrary quorum slices compactly

Instead, use k-of-n configuration that can recurse twice

- E.g., allows policies like 51% of each organization for 3/4 of organizations

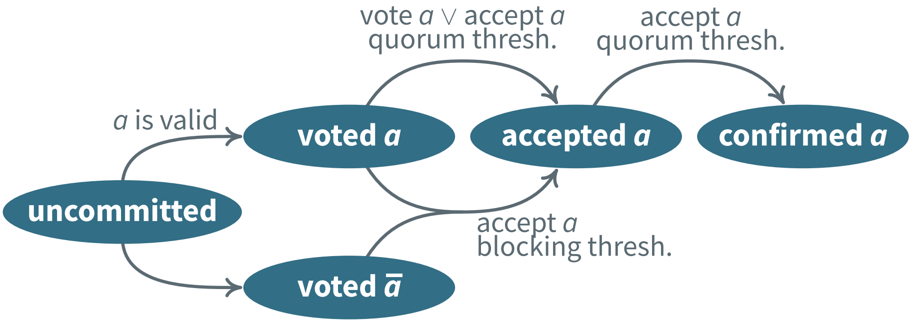
Vote messages

```
struct SCPStatement {
    PublicKey nodeID;      // v (node signing message)
    uint64 slotIndex;
    Hash quorumSetHash;
    union switch (SCPStatementType type) {
      case SCP_ST_PREPARE:
        SCPPrepare prepare;
      case SCP_ST_COMMIT:
        SCPCommit commit;
      case SCP_ST_EXTERNALIZE:
        SCPExternalize externalize;
      case SCP_ST_NOMINATE:
        SCPNominate nominate;
    } pledges;
};
struct SCPEnvelope {
    SCPStatement statement;
    Signature signature;
};
```

Transmit quorum slices as SHA-256 hash of SCPQuorumSet

- Use side protocol to request preimage if not cached

Main subroutine: federated voting



Nodes vote for or against a conceptual statement *a*

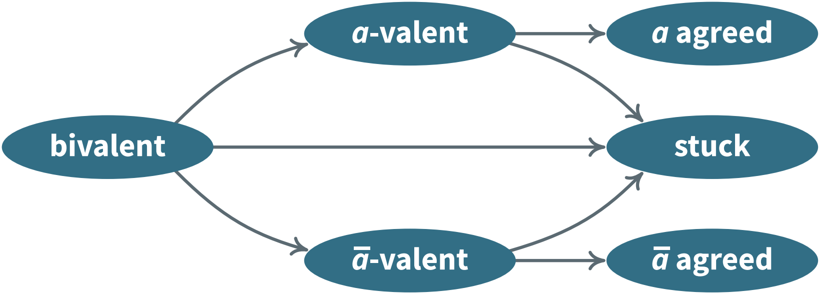
Can't *accept* contradictory statements if quorum intersection despite faulty nodes (intertwined) and in honest quorum (intact)

Can't *confirm* contradictory statements if intertwined

Could get stuck in *voted* or *accepted* stage

- But if one intact node *confirms* statement, all will

Federated voting outcomes



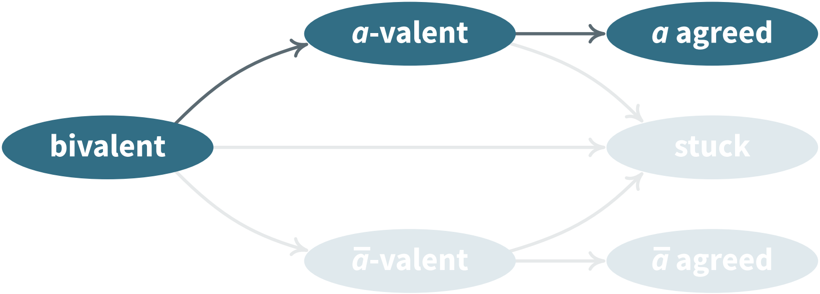
If you can vote for or against statement *a*, vote may get stuck

- E.g., split vote precludes quorum (since no way to change vote)
- Or was quorum but nodes failed before everyone learned of it

If you can't vote against *a*, then vote can always terminate

- As long as there's a non-failed quorum, it can always vote for *a*
- Call *a* irrefutable if honest nodes can't vote against it

Federated voting outcomes



If you can vote for or against statement *a*, vote may get stuck

- E.g., split vote precludes quorum (since no way to change vote)
- Or was quorum but nodes failed before everyone learned of it

If you can't vote against *a*, then vote can always terminate

- As long as there's a non-failed quorum, it can always vote for *a*
- Call *a* irrefutable if honest nodes can't vote against it

SCP nomination message

```
typedef opaque Value<>;

struct SCPNominate {
  Value voted<>;     // vote to nominate these values
  Value accepted<>;  // assert that these are accepted
};

union SCPStatement switch (SCPStatementType type) {
  case SCP_ST_NOMINATE:
    SCPNominate nominate;
  /* ... */
};
```

Nodes broadcast nominated values in `voted`

- Initially vote values in all received votes (ignoring optimization here)

Upon accepting nomination of a**, move from** `voted` **to** `accepted`

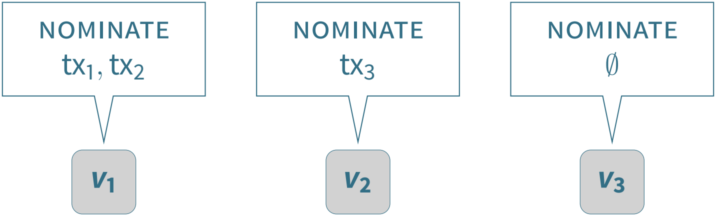
Stop voting for new values once any is confirmed nominated

- But continue accepting and repeating votes already cast

New: stop sending `SCPNominate` **when ballot confirmed prepared**

- Means NOMINATION phase overlaps with PREPARE phase

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

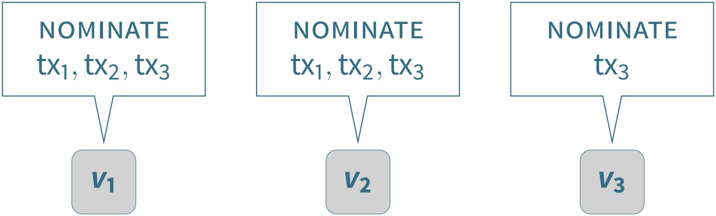
Nomination irrefutable, so will converge on set of values

Deterministically combine nominations into *composite* value x

Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

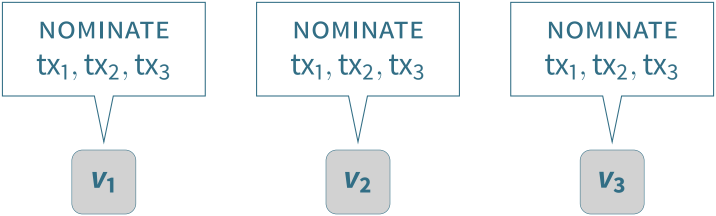
Nomination irrefutable, so will converge on set of values

Deterministically combine nominations into *composite* value x

Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

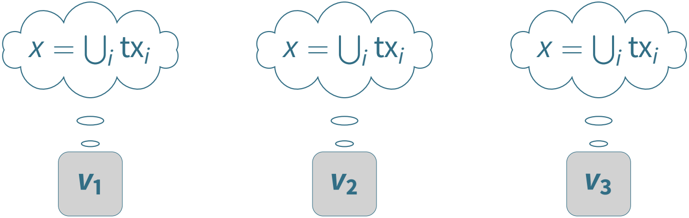
Nomination irrefutable, so will converge on set of values

Deterministically combine nominations into *composite* value x

Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

Nomination flow



Nodes nominate values and re-nominate any nominations seen

Stop adding to `votes` once any value confirmed nominated

Nomination irrefutable, so will converge on set of values

Deterministically combine nominations into *composite* value x

Nodes guaranteed to converge on same value x

- Complication: impossible to know when protocol has converged [FLP]
- c.f. asynchronous reliable broadcast

SCP ballots

```
struct SCPBallot {
  uint32 counter;            // n
  Value value;               // x
};
```

Composite nomination output must be run through balloting

- Guarantees safety even if started before nomination converges

A ballot b is a pair $\langle b.\text{counter}, b.\text{value}\rangle$ where $b.\text{counter}$ is a candidate output value

- Ballots totally ordered with `counter` more significant than `value`
- Nodes may vote to commit or abort a ballot, not both
- If a node confirms commit b for any b, it outputs $b.\text{value}$

Let prepared$(b) = \{\textbf{abort}\ b' \mid b' < b\ \textbf{ and }\ b'.\text{value} \neq b.\text{value}\}$

Invariant: cannot vote commit b unless federated voting has confirmed every statement in prepared(b)

SCP prepare message

```
struct SCPPrepare {
  SCPBallot ballot;
  SCPBallot *prepared;
  SCPBallot *preparedPrime;
  uint32 hCounter;
  uint32 cCounter;
};
```

vote-or-accept prepare(`ballot`)

if `prepared` $\neq$ **NULL: accept prepare**(`*prepared`)

if `preparedPrime` $\neq$ **NULL: accept prepare**(`*preparedPrime`)

if hCounter $\neq 0$**: confirm prepare**($\langle$**hCounter**, **ballot.value**$\rangle$)

if cCounter $\neq 0$**:**
$\{$**vote commit**($\langle n$, **ballot.value**$\rangle$) $\mid$ **cCounter** $\leq n \leq$ **hCounter**$\}$

Progress to COMMIT phase upon accepting commit of any ballot

Setting the prepare fields

`ballot.counter` starts at 1, increases w. timeouts and received messages (details in a few slides)

`ballot.value` b.value from highest b with confirmed prepared(b) (if any), otherwise composite nomination value

`prepared` highest b for which sender accepted prepared(b)

`prepared`$'$ highest b with accepted prepared(b) and different x from `prepared`

`hCounter` h.counter from highest h with confirmed prepared(h) and b.value $==$ h.value (new), else 0

`cCounter` 0 if `hCounter` $==$ 0 or internal "commit ballot" $c ==$ NULL. Else, c.counter. Note $c \leftarrow$ `ballot` when confirmed prepared and NULL when accepted aborted.

SCP commit message

```
struct SCPCommit {
    SCPBallot ballot;
    uint32 preparedCounter;
    uint32 hCounter;
    uint32 cCounter;
};
```

$\{$**accept commit**$(\langle n, \texttt{ballot.value}\rangle) \mid \texttt{hCounter} \leq n \leq \texttt{cCounter}\}$

vote-or-accept prepare$(\langle \infty, \texttt{ballot.value}\rangle)$

accept prepare$(\langle \texttt{preparedCounter}, \texttt{ballot.value}\rangle)$

confirm prepare$(\langle \texttt{hCounter}, \texttt{ballot.value}\rangle)$

$\{$**vote commit**$(\langle n, \texttt{ballot.value}\rangle) \mid n \geq \texttt{cCounter}\}$

SCP externalize message

```
struct SCPExternalize {
    SCPBallot commit;
    uint32 hCounter;
};
```

$\{\textbf{accept commit}(\langle n, \texttt{commit.value}\rangle) \mid \texttt{commit.counter} \leq n\}$

$\{\textbf{confirm commit}(\langle n, \texttt{commit.value}\rangle) \mid \texttt{commit.counter} \leq n \leq \texttt{hCounter}\}$

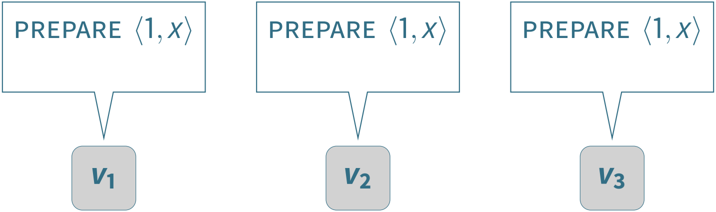
$\textbf{accept prepare}(\langle \infty, \texttt{commit.value}\rangle)$

$\textbf{confirm prepare}(\langle \texttt{hCounter}, \texttt{commit.value}\rangle)$

By the time you send this, already externalized `commit.value`

- Means you have confirmed committed a ballot with `commit`.*value*
- Goal is definitive record to help other nodes prove value/catch up

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

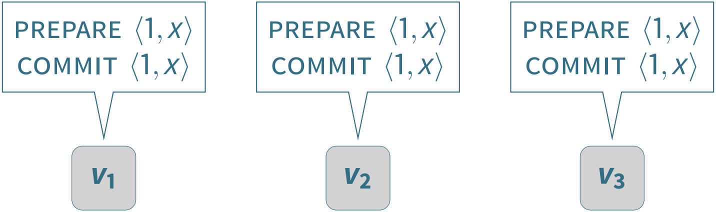
Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Nomination may finish converging in background

Or if any value confirmed prepared, all nodes will eventually see it confirmed prepared and start using that value

Balloting flow



In the common case, will prepare and commit nominated value

Else, arm timer when ballot counter reaches quorum threshold

Bump counter and restart with new ballot whenever

- Timer fires
- A blocking threshold is at a higher ballot counter

Nomination may finish converging in background

Or if any value confirmed prepared, all nodes will eventually see it confirmed prepared and start using that value

Questions?