An Algorithm for Managing Shared Teleconferencing State

Extremely Preliminary!

Scott Shenker
Abel Weinrib
Eve Schooler

IETF November 1993

Teleconferences

Shared State:

- Membership
- Roles (speaker, next-in-line, chairperson)
- Media encodings, encryption keys, etc.
- Payment model

Human endeavor:

- Deciding on shared state is not just a technical problem
- Decision criterion: policy issue (social/political)
- Means to implement: technical issue

Many different policy options:

- Anarchy, dictatorship, majority rule, ...

Telephone Examples

Two-party phone call:

- No change to membership possible
- Tight control of membership

Party Line:

- Anyone can join
- Loose control of membership

Typical teleconferencing mechanisms:

- Policy embedded in mechanism
- No policies in between the two extremes

Social Example

Country Club Membership:

- Need three sponsors to join
- Can be blackballed by a single member
- Membership policy can be changed by changing the charter of the country club
- General mechanism: voting at meetings

Typical social institutions:

- Intermediate membership policies
- Policy not embedded in mechanism

This is what we want to emulate!

Goals of this Work

1. Express policy

   - Define general family of policies
   - Explore the *tight/loose* spectrum

2. Implement policy

   - Define a single mechanism to implement this family of policies
   - Variant of two-phase commit
     - A member initiates change in shared state
     - Propose > Collect Votes > Announce Results
     - Members exchange messages for voting, setting locks, committing, etc.

Context and Notation

Assume (for now) that

- Messaging is reliable but asynchronous
- All messages from a particular sender to a particular receiver arrive in order

Notation

- $\mathcal{S}$: shared state
  - S_i state variables
  - $S_1 = M$ membership
  - $S_2 = P$ session policies

- $\mathcal{O}_i$: operations on state variable S_i
  - O_i particular operation in $\mathcal{O}_i$
  - General change operation $op = O_i \ldots \circ O_j$
  - $STATE(op)$: all possible state variables changed by op

Expressing Policy

Three dimensions to policy

- Voting policies
- Consistency policies
- Initiator policies

Voting Policies

For every O_i there is some *voting rule* $V(O_i)$

- Takes set of votes {YES, NO, ABSTAIN}
- Returns 0 (Fail) or 1 (Pass)
- Depends on operation, not just on i
- Independent of initiator

For a general change $op = O_i \ldots \circ O_j$

- $V(op) = V(O_i) \ldots \circ V(O_j)$

Null voting rules:

- No votes (other than the initiator's) need be cast

Consistency Policies

Views of shared state can be inconsistent

- Permanent: operations executed in different order
  - Long-lived inconsistency can render teleconference inoperable
- Temporary: not all change messages have been delivered (inevitable)
  - Votes can be cast with different views of shared state

Two Consistency Conditions

Eventual Consistency in E

- Eventual set $E \subseteq \mathcal{S}$
- When session is quiescent, all members have identical views of the state variables in E.

Voting Consistency

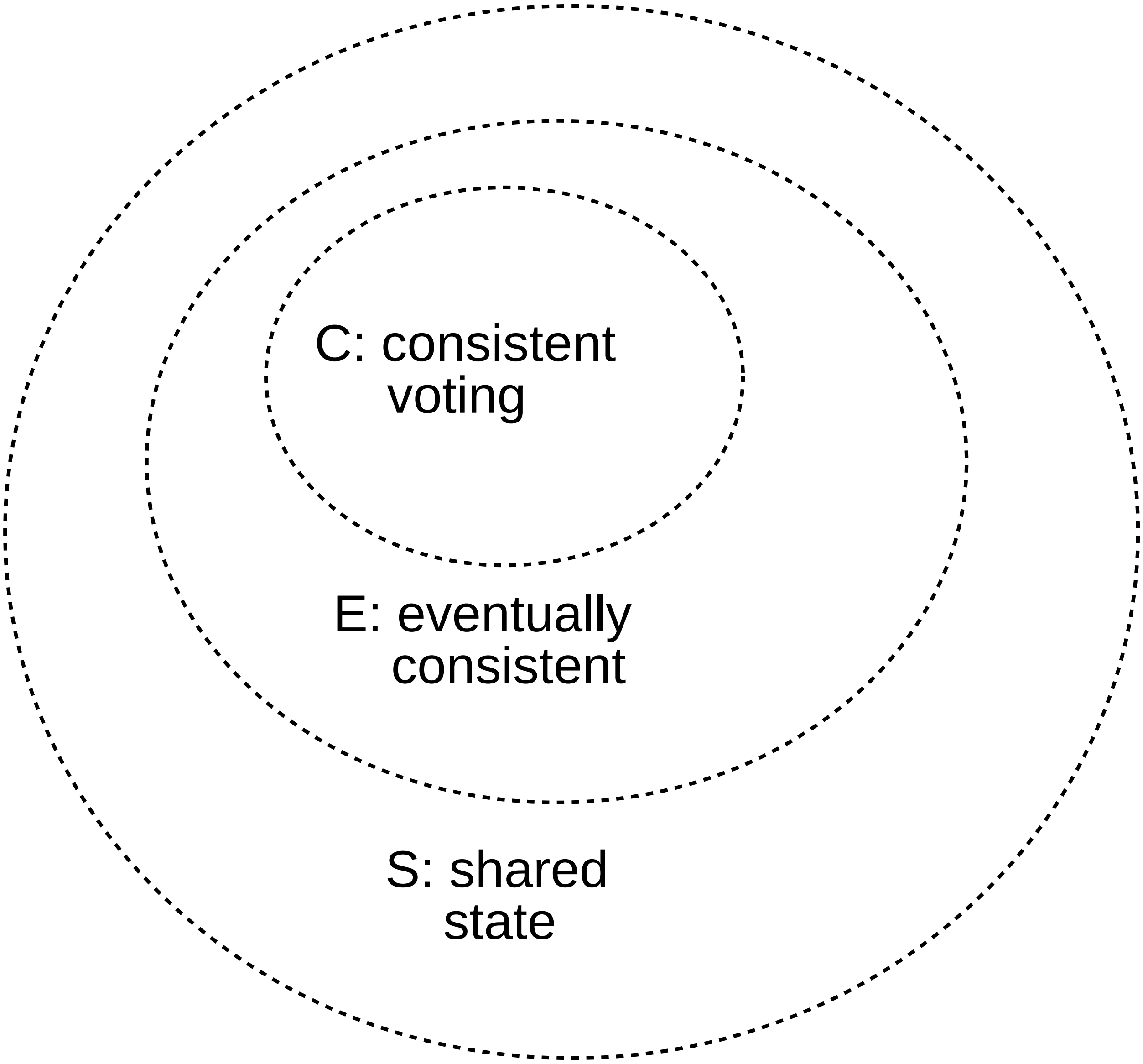
- *Critical* state C and *Consistent* state I
- All votes cast on changes in I are made with consistent views of C ($\Rightarrow C \subseteq E$)

Choice of I, C, E is statement of consistency policy!

Restrictions

- $I \subseteq E$
- $I = C$ *(will discuss later)*
- Voting consistency independent of operation

Sets


C: consistent
voting
E: eventually
consistent
S: shared
state

Initiation Policies

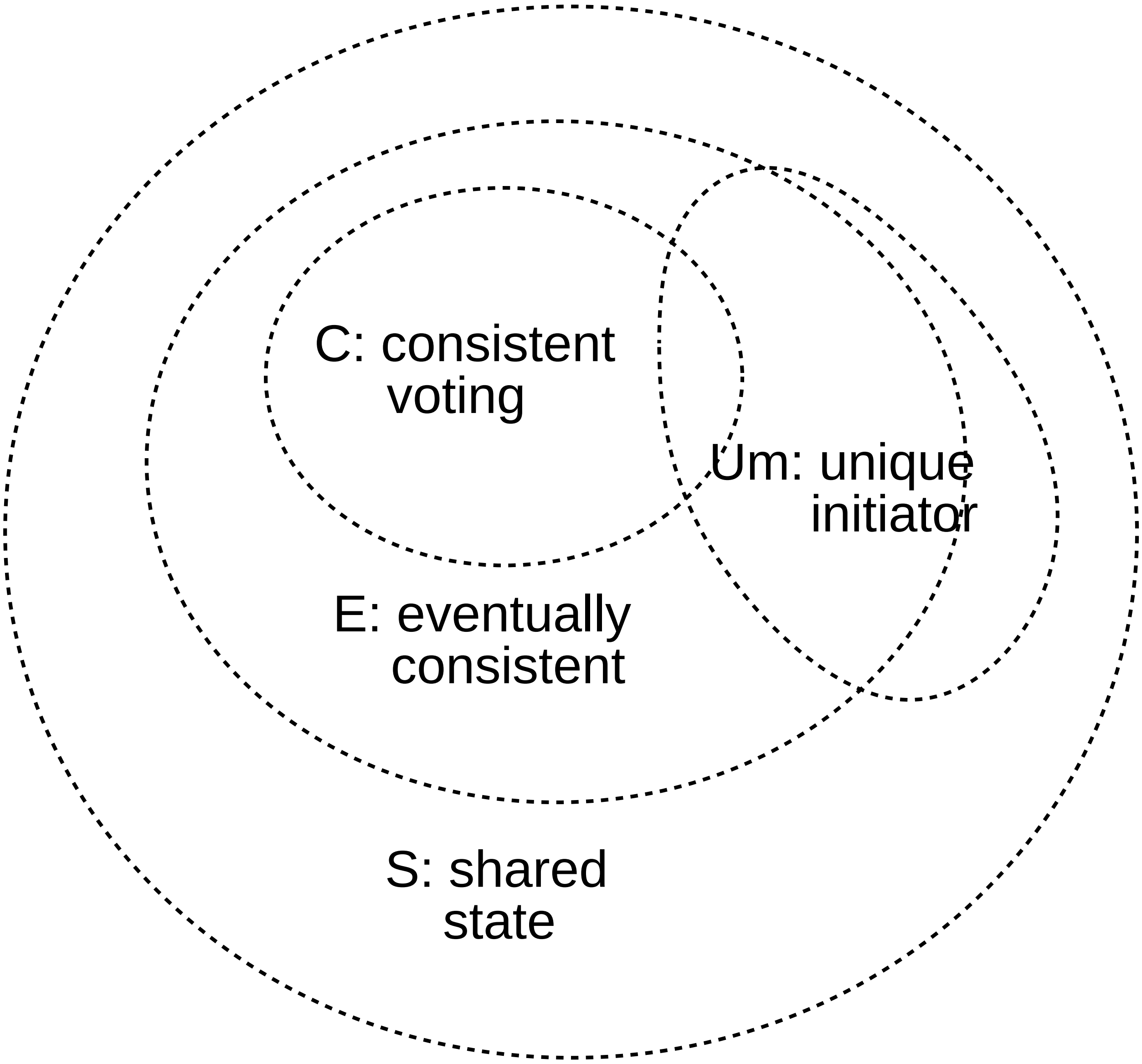
Can have general set of policies about which members can initiate changes to various state variables.

These policies define set U_m for each member m

- Changes to variables in U_m can only be initiated by m
- $U_m \cap U_n = \phi$ when $m \neq m$
- $U = \cup_m U_m$
- *Paper wrong*

Unique initiation does not imply unilateral control

Sets



Initiation policies are orthogonal to consistency policies!

Special Role of Membership

If membership is not in C, then we suggest that C is empty

- All members must have a consistent view of C when voting on changes in C
- If membership is not in C, what does "*all members*" mean?

For now, assume that membership is in C

Example

Shared State $\mathcal{S}$

- Membership
- Encryption key
- Common names of members
- Next-to-speak

Policy:

- Membership: in C
  - Adding member: unanimous vote
  - Deleting member: if and only if member wants to leave
- Encryption Key: in $E - C$
  - Null voting
- Common name of m: in $E - C$ and U_m
- Next-to-Speak: not in E
  - Majority rule

Any Questions?

Implementing Policy

Variant of two-phase commit

- Dynamic group membership
- Looser correctness conditions
- Spectrum of voting policies

Basic mechanism

- Change initiated by single member
- Propose > Collect Votes > Announce results
- Change assigned unique `id`
- Members are either busy (locked) or non-busy (unlocked)

Basic Message Functions:

- Poll, Lock, Respond, Announce

Message Types

`Poll(Id: id, Operation: op)`

- Requests vote on proposed operation
- Response is {YES, NO, ABSTAIN}

`Lock(Id: id)`

- Requests exclusive lock
- Response is {OK, BUSY}

`Poll_Lock(Id: id, Operation: op)`

- Requests exclusive lock and vote on proposed operation
- Response is {YES, NO, ABSTAIN, BUSY}

`Response(Id: id, Response: response)`

- Response to `Poll`, `Lock`, or `Poll_Lock` message
- No response

`Announce(Id: id, Operation: op)`

- Operation op is to be applied to state
- No response

`Release(Id: id, Operation: op)`

- Associated lock is released
- No response

`Commit(Id: id, Operation: op)`

- Operation op is to be applied to state
- Associated lock is released
- No response

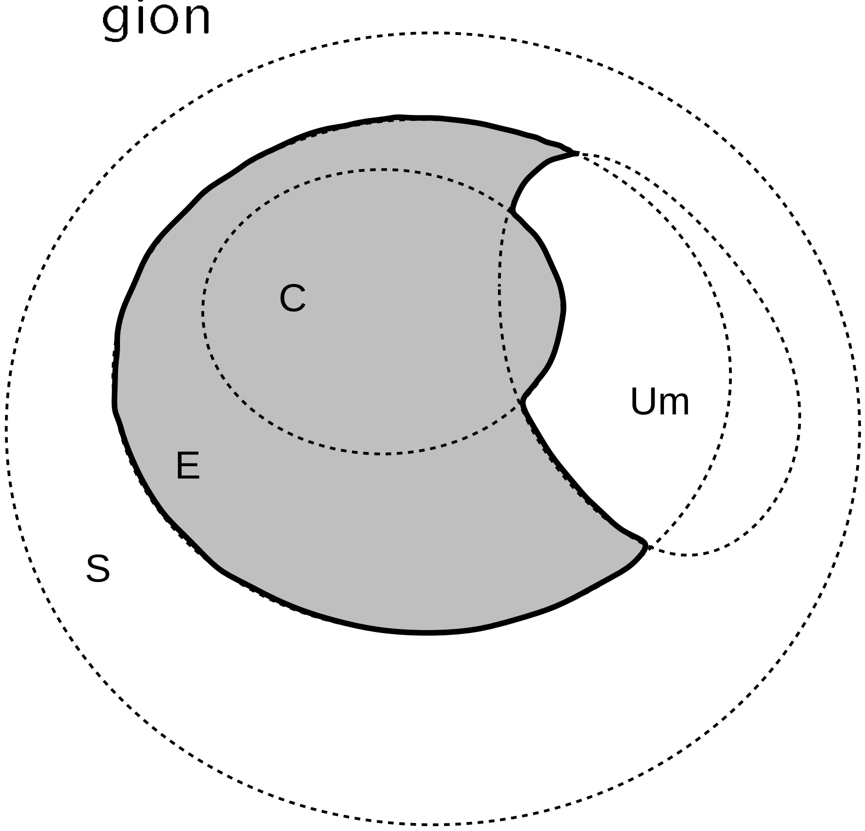
Message Exchanges

Four Basic Exchange Patterns:

- `Poll_Lock` > `Response` > `Commit` or `Release`
- `Lock` > `Response` > `Commit` or `Release`
- `Poll` > `Response` > `Announce`
- `Announce`

Cross Product of Polling and Locking

- Poll: if $V(op)$ is not null
- Lock: if $STATE(op)$ intersects shaded region



Message Exchanges

Consider some change operation op initiated by member m

Message exhange depends on:

- Does it require a vote?
- $STATE(op) \cap C \neq \phi$?
- $STATE(op) \cap E - C \neq \phi$?
- $STATE(op) \subseteq U_m$?

VOTE and $STATE(op) \cap C \neq \phi$:

- `Poll_Lock` $>$ `Response` $>$ `Commit` or `Release`
- Release is sent if vote fails or if any BUSY response is received
- Locking needed to ensure no two operations are going on simultaneously

Same sequence applies for:

- VOTE, $STATE(op) \cap C = \phi$, $STATE(op) \cap E - C \neq \phi$, $STATE(op) \notin U_m$
- Locking needed to ensure operations are done in same order

NO_VOTE and $STATE(op) \cap C \neq \phi$:

- `Lock` $>$ `Response` $>$ `Commit` or `Release`
- Release is sent if any BUSY response is received

Same sequence applies for:

- NO_VOTE, $STATE(op) \cap C = \phi$, $STATE(op) \cap E - C \neq \phi$, $STATE(op) \notin U_m$

VOTE, $STATE(op) \cap C = \phi$, $STATE(op) \cap E - C \neq \phi$, $STATE(op) \in U_m$:

- `Poll` $>$ `Response` $>$ `Announce`
- No lock needed because unique initiator ensures no variables will be changed by any other operation

Same sequence applies for:

- VOTE, $STATE(op) \subseteq \mathcal{S} - E$

NO_VOTE, $STATE(op) \cap C = \phi$, $STATE(op) \cap E - C \neq \phi$, $STATE(op) \in U_m$:

- `Announce`
- Unique initiator means eventual consistency is assured without locks.

Same sequence applies for:

- NO_VOTE, $STATE(op) \subseteq \mathcal{S} - E$
- No voting and no consistency conditions allow a unilateral announcing of the new state variables.

Consistency Conditions

Eventual Consistency Whenever there are no outstanding messages, all participants have the same view of E.

Consistent Voting All votes cast involving changes to C are done with identical views of C.

Loose Control of Membership

What happens when membership is not in C?

- Previously argued that C must be empty
- If E empty, everything vacuously OK

For nonempty E

- Our mechanism does not work
- $E \subseteq U$: use periodic `Announces`
  - changes still use `Poll` messages
- Xerox Clearinghouse algorithms
  - after-the-fact timestamp ordering
  - need to keep history on membership
- Both of the above need some method to eventually reach entire membership

Unreliable Transport

Design Choice

- Cope with inconsistencies rather than achieve correctness
- Need hooks for resynchronizing state

Unilateral withdrawal: when all else fails...

- Send `Announce` declaring departure
- Modified consistent voting condition
- Voting is not consistent but *plausible*

Open Issues

- Finer grain concurrency control
- Critical and noncritical membership
- Session initiation
- Voting by incoming members
- Changes to policy
- Responses to protocol errors
- Better and more complete formalism
- **Is this useful in building real systems?**