

CS603: Distributed Systems

Lecture 10: Membership, Reliable Ordered Multicast

Membership Service

- Needed for distributed protocols that require knowledge of alive processes
- Static: list is known before, track processes that crash
- Dynamic: processes can join, leave and crash
- Need to detect failures (we know we can not do it accurately)
- Need to agree on the current list of processes

A Membership Protocol

- One of the processes (oldest) will act as coordinator
- Each process maintains a list with alive processes (list has to be the same) WHY?
- All processes track each other (ping or I am alive)
- If timeout occurs - process that did not answered is considered crashed, he will have to rejoin with another identifier
- Two cases:
 - coordinator is alive (normal-case)
 - coordinator fails

Normal-Case

- Coordinator detects a failure or receives a join, he starts a two-phase protocol to ensure the list of alive members is updated consistently
- **Phase 1**: coordinator sends all add and delete event to everybody
- Every process acknowledges
- Coordinator must wait for a majority of acknowledges
- **Phase 2**: If coordinator receives majority, then sends the modifications (may include any failure detected during first phase)

Coordinator Fails: 3 Phases Protocol

- If a process detects that coordinator failed, second process on the list becomes coordinator
- **Phase 1**: new coordinator informs the other processes that coordinator has failed, asks for pending add/delete operations, collects acknowledgments and current membership information
- **Phase 2 and 3** similar with normal case

Reliable Multicast

Unicast, Broadcast, Multicast

- **Unicast**: Messages are sent from exactly one process to one process
- **Broadcast**: Messages are sent from exactly one process to all processes on the network
- **Multicast**: Messages are sent from exactly one process to several processes (**referred as group**) on the network

Reliable Communication

- Unicast: one sender and one receiver
- Multicast: one sender and many receivers
- Reliable unicast: guarantees delivery of messages, if the other party fails, there is no service
- What is the meaning of reliable multicast in the context of process failures?

Naïve Approach

- Use a reliable one-to-one send operation:
- A basic multicast primitive guarantees a correct (non-faulty) **process will eventually deliver the message, as long as the sender does not crash.**
- What if the sender crashes after he sent the message and some processes received the message and some other did not?

Meaning of Reliability in Multicast

- **Integrity**: A correct process p delivers a message m at most once.
- **Validity**: If a correct process multicasts message m , then it will eventually deliver m .
- **Agreement**: If a correct process delivers message m , then all the other correct processes in the group will eventually deliver m .

Ordered Multicast

- **FIFO ordering**: If a correct process multicasts m and then multicasts m' , then every correct process that delivers m' will have already delivered m .
- **Causal ordering**: If multicast $m \rightarrow$ multicast m' then any correct process that delivers m' will have already delivered m .
- **Total ordering**: If a correct process delivers message m before m' , then any other correct process that delivers m' will have already delivered m .

ISIS

- Toolkit for distributed programming
- Useful for managing replicated data, synchronizing distributed computations, automating recovery, and dynamically reconfiguring a system to accommodate changing workloads
- Developed at Cornell

ISIS Total Ordered Multicast

- Uses sequences associate with each message and ID of processes to determine order
- Each process maintains a queue with messages received
- Messages can be ready to deliver or not based on what a process know about what other processes did (the sequence)

ISIS Total Ordered Multicast (cont)

- Sender multicasts the message to everyone
- Upon receiving a message M each receiver R_i
 1. Adds M to the queue
 2. Marks the message *undeliverable*
 3. Sends ack to the sender with a sequence number seq that is *the latest sequence number received* + 1, suffixed with the R_i 's ID.
- Sender collects all acks from the receivers
 1. calculates $\text{final_seq} = \text{maximum}(\{seq_{ij}\})$
 2. multicasts final_seq to all processes
- Upon receiving final seq each receiver
 1. marks M as *deliverable*,
 2. reorders the queue based on seq
 3. delivers the set of messages with lower seq and marked as *deliverable*.

TOTEM: The single-ring protocol

- Uses a *circulating token* containing among others:
 - A *seq* field with the sequence number of the last message that was sent
 - An *aru* field with the sequence number of the last message that has been received by all processors
- Only the processor that holds the token can send a message

Safe Delivery

- Consistent with Total/Agreed order.
- Message is delivered after received by all processors.

TOTEM: The single-ring protocol (II)

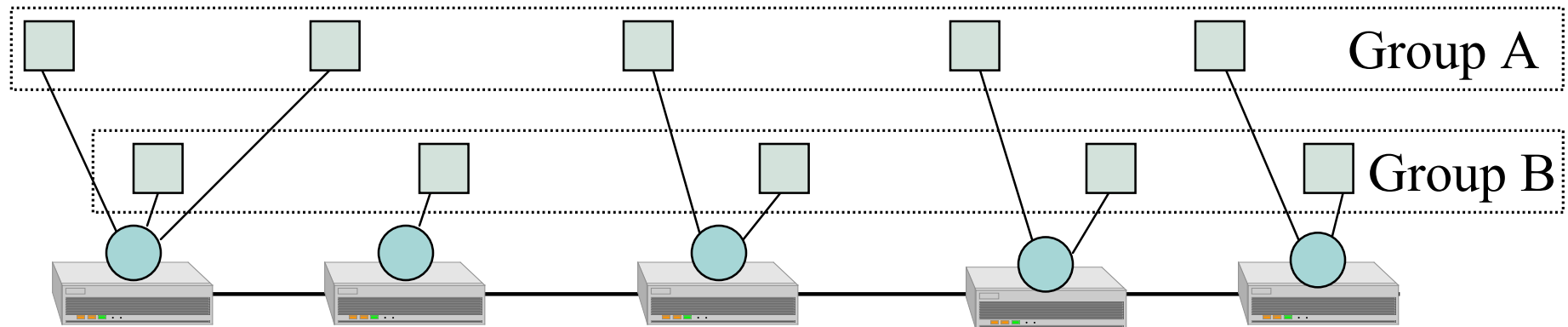
- *aru* field used to implement safe delivery:
 - Tells processors which messages have been received by every processor in the ring
- Token also provides information about the aggregate message backlog of the processors on the ring
 - Results in a fairer bandwidth allocation among processors

Membership and Reliable Multicast

- Message delivery
- Group membership changes
- They are interleaved
- Does this matter?

Group Communication Systems

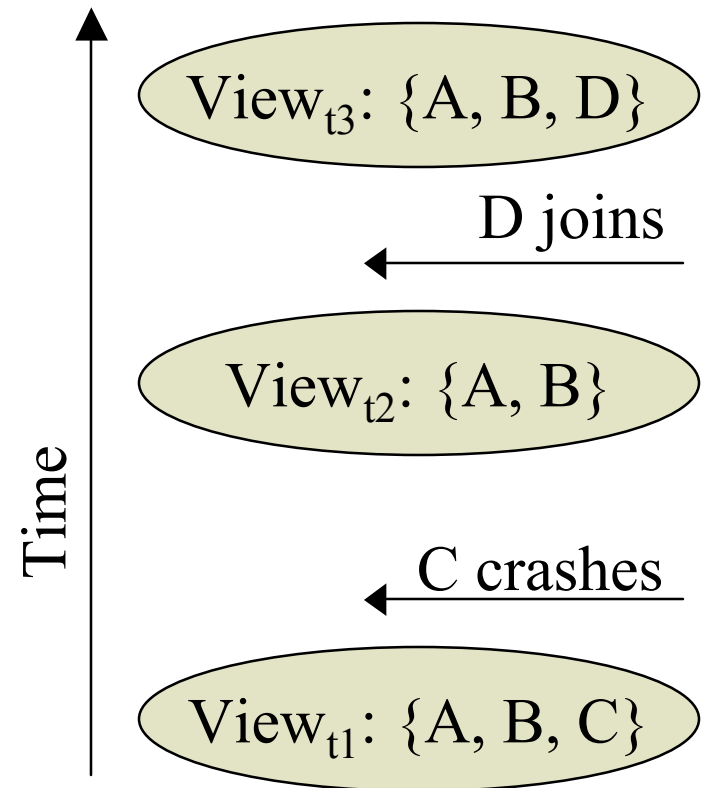
- ◆ Collaborative applications: computing, white-boards, video-conference
- ◆ Distributed transactions and database replication
- ◆ Cluster management and monitoring
- ◆ Highly available servers



- Reliable and ordered message delivery (unicast and broadcast)
- Group membership service supporting **process failures, network partitions and merges**

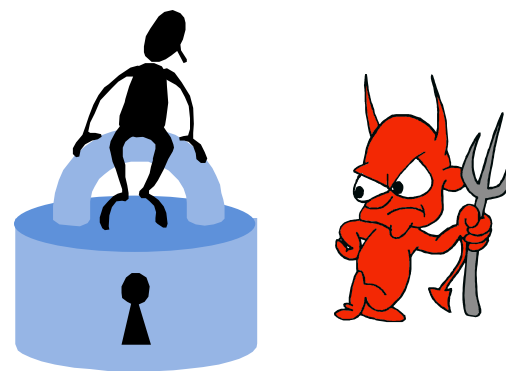
Group Communication Semantics

- **View**: list of group members at a certain time
- **Virtual Synchrony Model (VS)** : all the messages are delivered in the group view they were sent in (*Sending View Delivery*).
- **Extended Virtual Synchrony Model (EVS)**: all the messages are delivered in the same group view. This may be different from the group view the message was sent in (*Same View Delivery*).



Example: Data Integrity and Confidentiality

- Requires a shared key between members of a group
- Can be used to bootstrap other group security services (e.g. confidentiality and integrity): **key management** is a critical building block
- Key must be changed when the group changes to restrict data access only to current group members

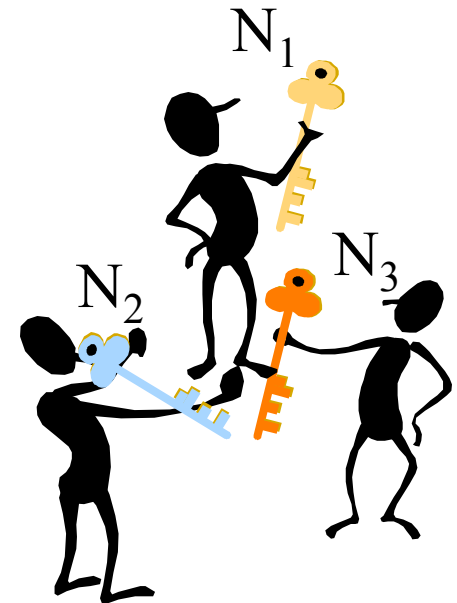


Group Diffie-Hellman (GDH)

- Contributory group key agreement protocol suite based on Diffie-Hellman key exchange.
- Each member uses a list of partial keys $g_i^{\frac{N_1 * \dots * N_n}{N_i}} \bmod p$

$$K_{group} = g^{N_1 * \dots * N_n} \bmod p$$

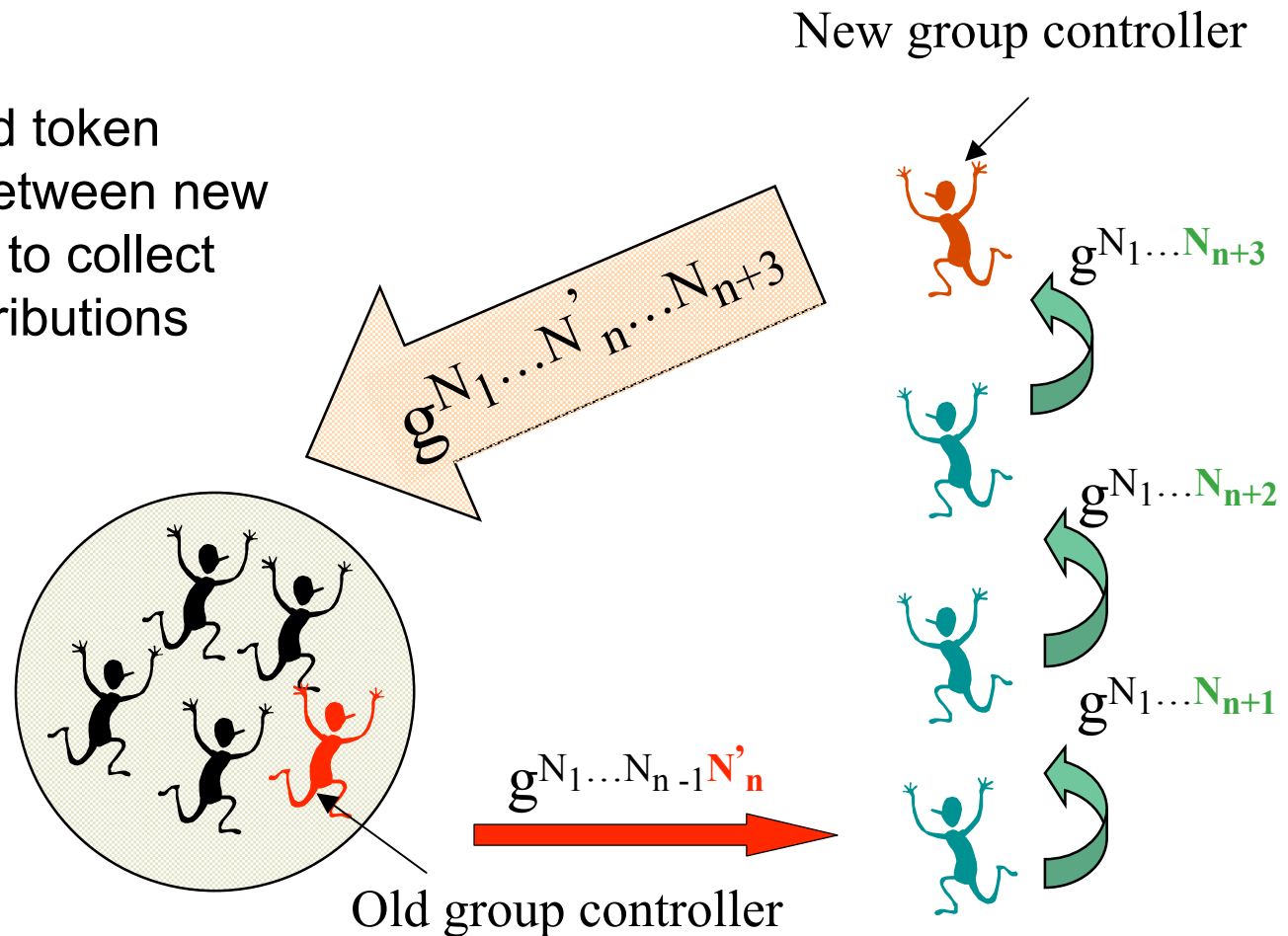
- A member of the group (**group controller**) is maintaining the list.
- Messages are authenticated by means of digital signatures.



GDH Merge

STEP 1

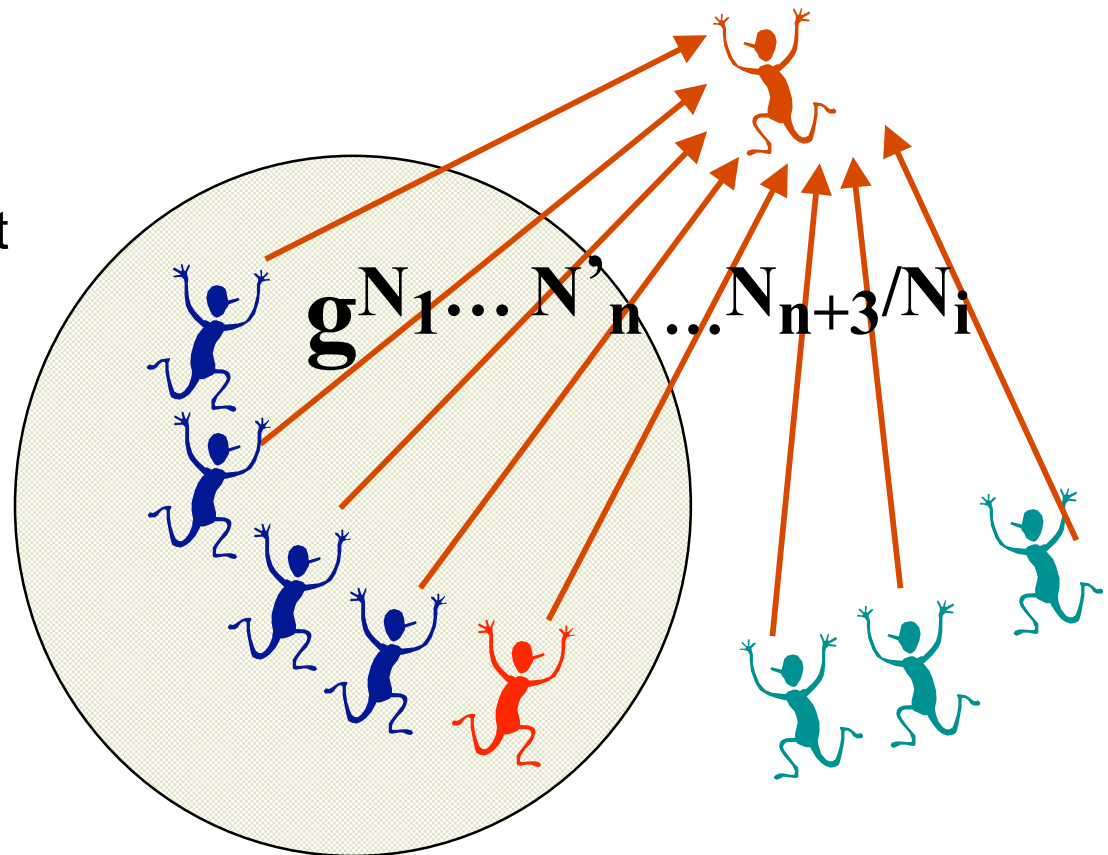
- Refreshed token passed between new members to collect their contributions



GDH Merge (cont.)

STEP 2

- Old and new members factor out their share and send it to the new controller.

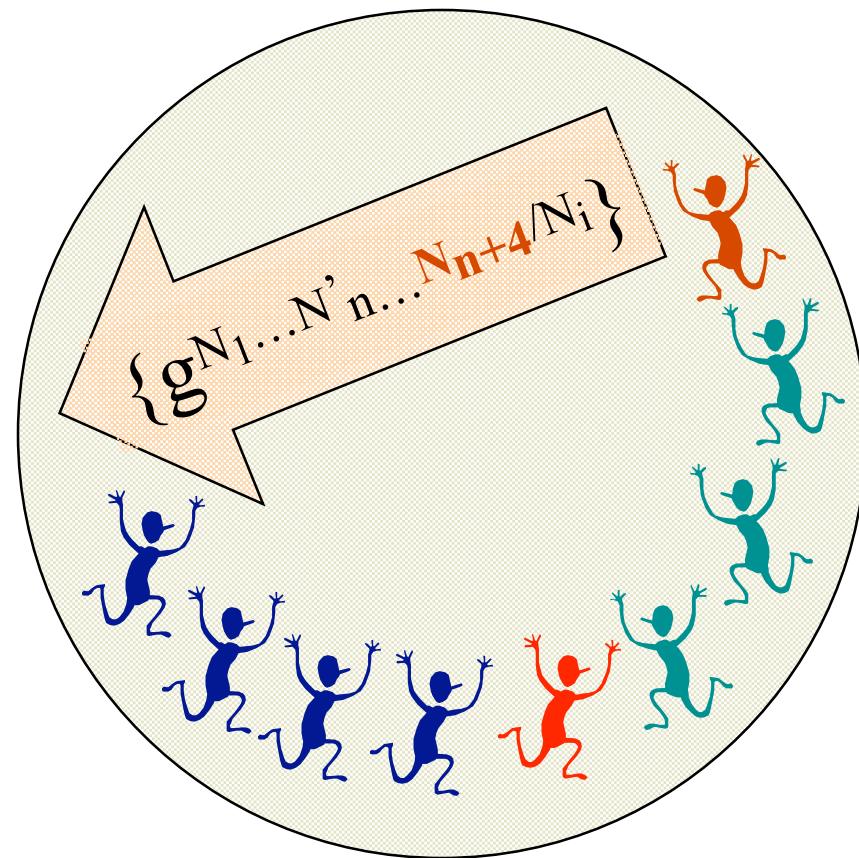


GDH Merge (cont.)

STEP 3

- New controller collects factored out tokens in a key list, adds its share to the key list, and broadcasts the list to the new group.

NOT ATOMIC !

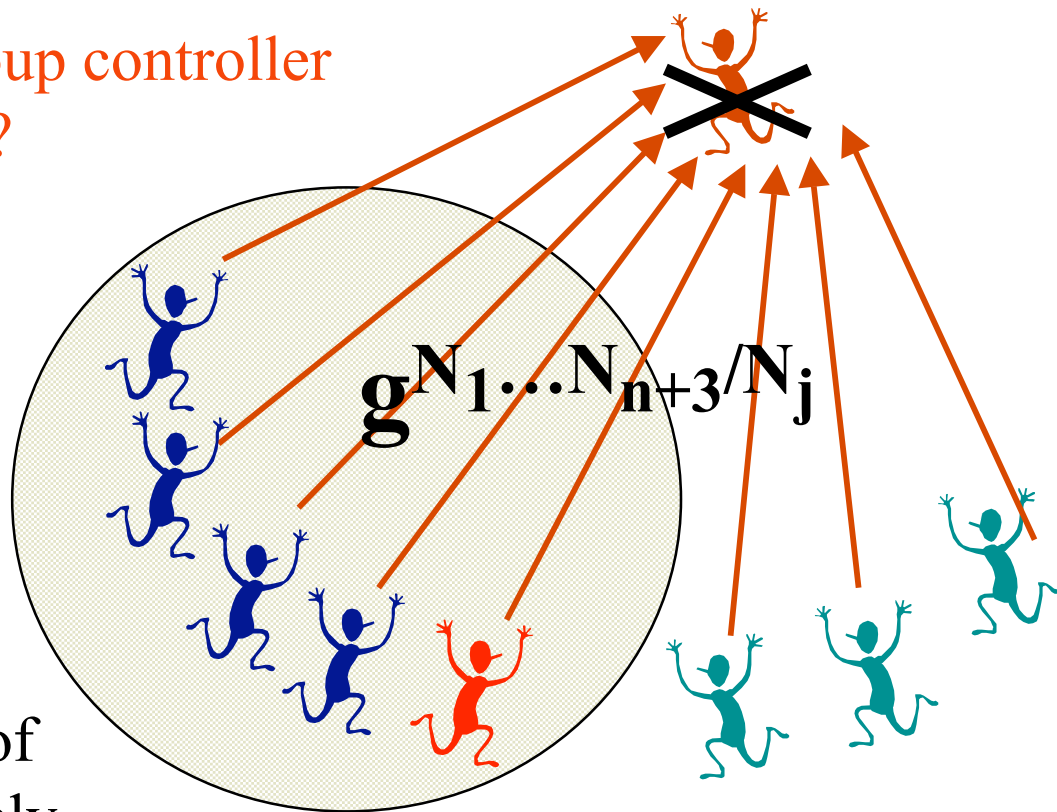


Fault Scenario

WHAT IF the new group controller
'crashes' at this point ?

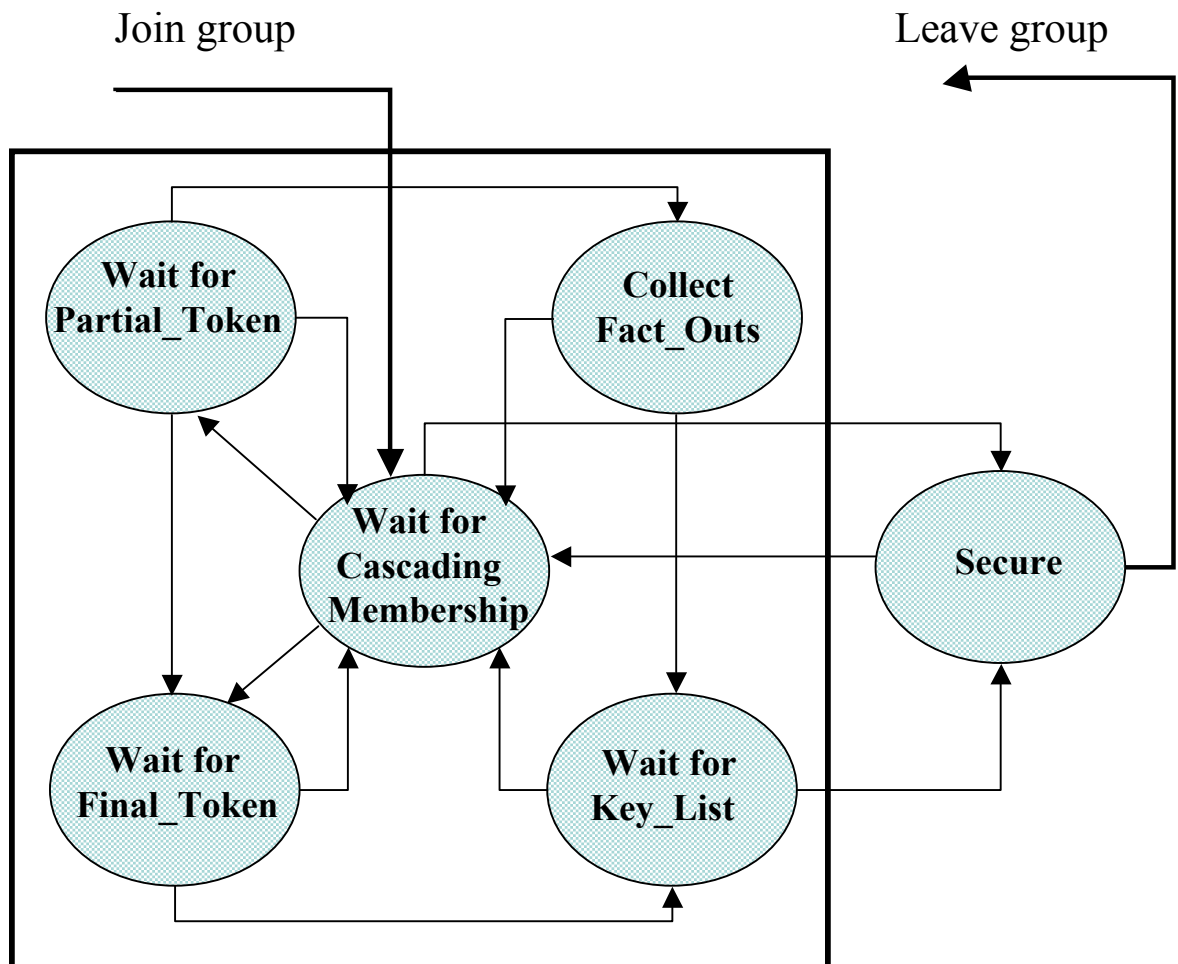
The protocol blocks.

The protocol must
handle any sequence of
group changes (possibly
cascading).



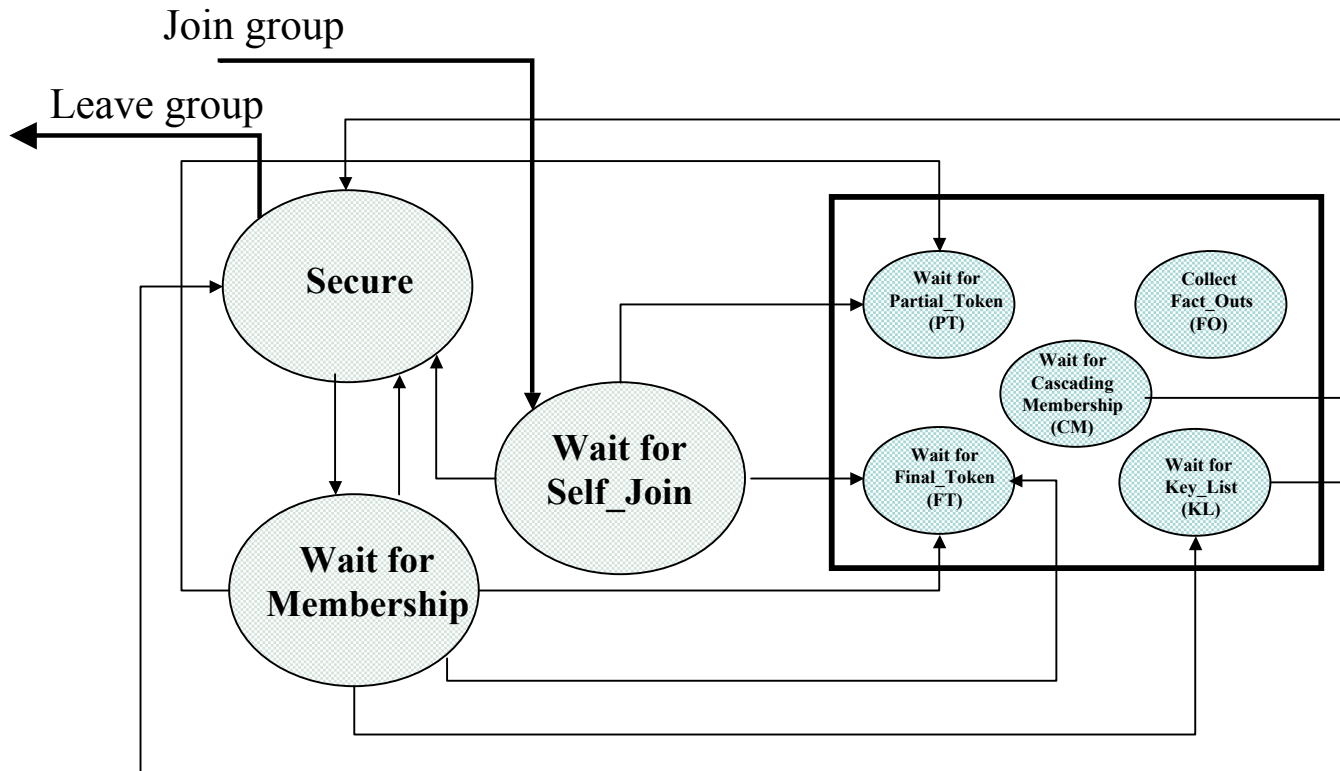
Basic Algorithm

- Uses membership service to make consistent decisions
- Total order delivery service used to ensure correctness



A process can crash/disconnect in any state.
The network can partition/merge in any state.

Optimized Algorithm



Reduces the cost to about 50% if no cascaded event happens.

A process can crash/disconnect in any state.
The network can partition/merge in any state.

So we got the group key...

- We can use it to encrypt communication in the group
- Sending View Delivery Property will ensure that NO MEMBER CAN RECEIVE A MESSAGE ENCRYPTED WITH OTHER KEYS THAN THE CURRENT KEY
- What happens in the EVS case?

What is the cost?

- When a membership changes all processes must agree that they are not going to send any more messages before changing the list/view/membership
- Then they block, no message are sent while the processes are changing the membership and computing a new key