CS 194: Distributed Systems
Process resilience, Reliable Group Communication

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

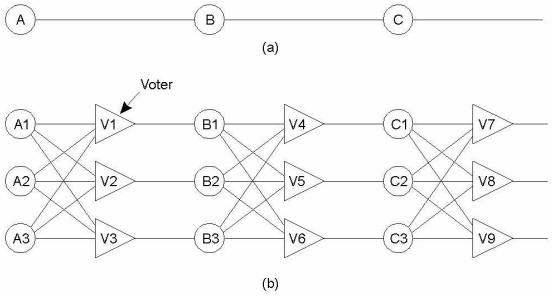
Some definitions…

- **Availability**: probability the system operates correctly at any given moment
- **Reliability**: ability to run correctly for a long interval of time
- **Safety**: failure to operate correctly does not lead to catastrophic failures
- **Maintainability**: ability to "easily" repair a failed system

… and Some More Definitions (Failure Models)

- **Crash failure:** a server halts, but works correctly until it halts
- **Omission failure**: a server fails to respond to a request
- **Timing failure**: a server response exceeds specified time interval
- **Response failure**: server's response is incorrect
- **Arbitrary (Byzantine) failure**: server produces arbitrary response at arbitrary times

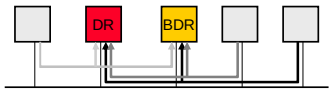
Masking Failures: Redundancy



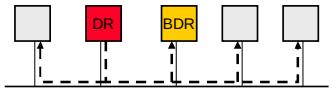
- How many failures can this design tolerate?

Example: Open Shortest Path First (OSPF) over Broadcast Networks

1) Each node sends an route advertisements to multicast group *DR-rtrs*
   - Both designated router (DR) and backup designated router (BDR) subscribe to this group



2) DR floods route advertisements back to all routers
   - Send to *all-rtrs* multicast group to which all nodes subscribe

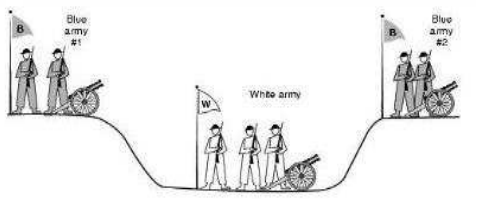


Agreement in Faulty Systems

- Many things can go wrong…
- Communication
  - Message transmission can be unreliable
  - Time taken to deliver a message is unbounded
  - Adversary can intercept messages
- Processes
  - Can fail or team up to produce wrong results
- Agreement very hard, sometime impossible, to achieve!

Two-Army Problem

- "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."



- What is the solution?

7

Byzantine Agreement [Lamport et al. (1982)]

- Goal:
  - Each process learn the true values sent by correct processes
- Assumptions:
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded

8

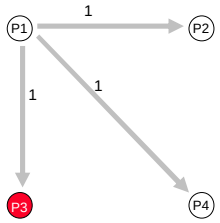
Byzantine Agreement Result

- In a system with m faulty processes agreement can be achieved only if there are $2m+1$ functioning correctly
- Note: This result only guarantees that each process receives the true values sent by correct processors, but it does not identify the correct processes!

9

Byzantine General Problem: Example

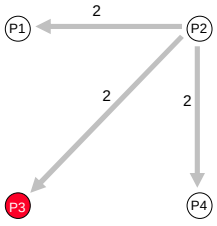
- Phase 1: Generals announce their troop strengths to each other



10

Byzantine General Problem: Example

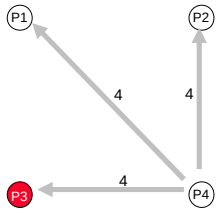
- Phase 1: Generals announce their troop strengths to each other



11

Byzantine General Problem: Example

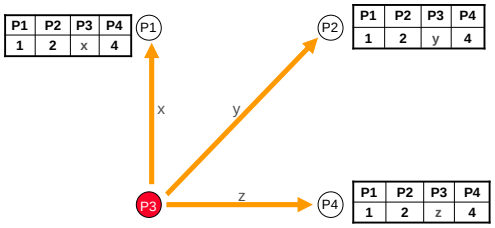
- Phase 1: Generals announce their troop strengths to each other



12

Byzantine General Problem: Example

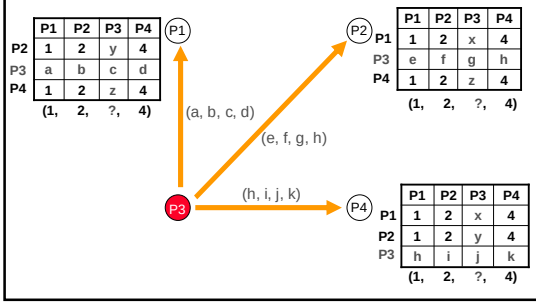
- Phase 2: Each general construct a vector with all troops



13

Byzantine General Problem: Example

- Phase 3: Generals send their vectors to each other and compute majority voting



14

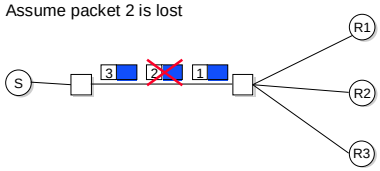
Reliable Group Communication

- **Reliable multicast**: all nonfaulty processes which do not join/leave during communication receive the message
- **Atomic multicast**: all messages are delivered in the same order to all processes

15

Reliable multicast: (N)ACK Implosion

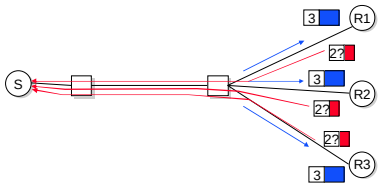
- (Positive) acknowledgements
  - Ack every n received packets
  - What happens for multicast?
- Negative acknowledgements
  - Only ack when data is lost
  - Assume packet 2 is lost



16

Reliable multicast: (N)ACK Implosion

- When a packet is lost all receivers in the sub-tree originated at the link where the packet is lost send NACKs



17

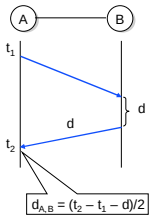
Scalable Reliable Multicast (SRM) [Floyd et al '95]

- Receivers use timers to send NACKS and retransmissions
  - Randomized: prevent implosion
  - Uses latency estimates
    - Short timer → cause duplicates when there is reordering
    - Long timer → causes excess delay
- Any node retransmits
  - Sender can use its bandwidth more efficiently
  - Overall group throughput is higher
- Duplicate NACK/retransmission suppression

18

Inter-node Latency Estimation

- Every node estimates latency to every other node
- Uses session reports
  - Assume symmetric latency
  - What happens when group becomes very large?



Repair Request Timer Randomization

- Chosen from the uniform distribution on

$$2^i[C_1 d_{S,A}, (C_1 + C_2) d_{S,A}]$$

  - A – node that lost the packet
  - S – source
  - C_1, C_2 – constants
  - $d_{S,A}$ – latency between source (S) and A
  - i – iteration of repair request tries seen
- Algorithm
  - Detect loss → set timer
  - Receive request for same data → cancel timer, set new timer
  - Timer expires → send repair request

Timer Randomization

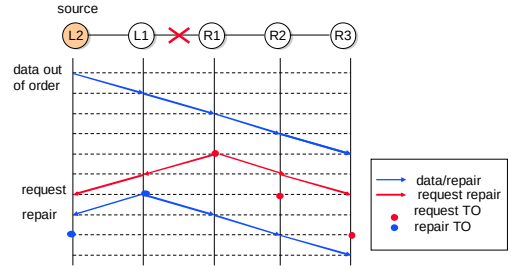
- Repair timer similar
  - Every node that receives repair request sets repair timer
  - Latency estimate is between node and node requesting repair
  - Use following formula:

$$2^i[D_1 d_{R,A}, (D_1 + D_2) d_{R,A}]$$

    - D_1, D_2 – constants
    - $d_{R,A}$ – latency between node requesting repair (R) and A
- Timer properties – minimize probability of duplicate packets
  - Reduce likelihood of implosion (duplicates still possible)
  - Reduce delay to repair

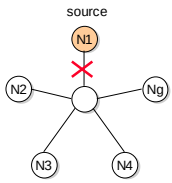
Chain Topology

- $C_1 = D_1 = 1$, $C_2 = D_2 = 0$
- All link distances are 1



Star Topology

- $C_1 = D_1 = 0$,
- Tradeoff between (1) number of requests and (2) time to receive the repair
- $C_2 <= 1$
  - E(# of requests) = g –1
- $C_2 > 1$
  - E(# of requests) = 1 + (g-2)/C_2
  - E(time until first timer expires) = 2C_2/g
- $C_2 = \sqrt{g}$
  - E(# of requests) = $\sqrt{g}$
  - E(time until first timer expires) = $1/\sqrt{g}$



Bounded Degree Tree

- Use both
  - Deterministic suppression (chain topology)
  - Probabilistic suppression (star topology)
- Large C_2/C_1 → fewer duplicate requests, but larger repair time
- Large C_1 → fewer duplicate requests
- Small C_1 → smaller repair time

Adaptive Timers

- C and D parameters depends on topology and congestion → choose adaptively
- After sending a request:
  - Decrease start of request timer interval
- Before each new request timer is set:
  - If requests sent in previous rounds, and any dup requests were from further away:
    - Decrease request timer interval
  - Else if average dup requests high:
    - Increase request timer interval
  - Else if average dup requests low and average request delay too high:
    - Decrease request timer interval

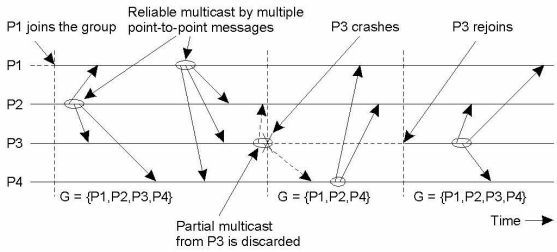
25

Atomic Multicast

- All messages are delivered in the same order to "all" processes
- **Group view**: the set of processes known by the sender when it multicast the message
- **Virtual synchronous multicast**: a message multicast to a group view G is delivered to all nonfaulty processes in G
  - If sender fails after sending the message, the message may be delivered to no one

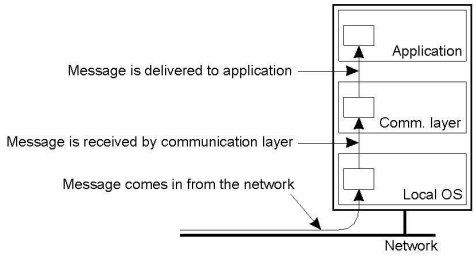
26

Virtual Synchronous Multicast



27

Virtual Synchrony Implementation [Birman et al., 1991]



- The logical organization of a distributed system to distinguish between message receipt and message delivery

28

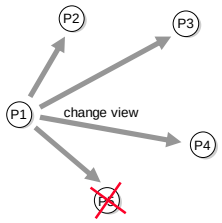
Virtual Synchrony Implementation: [Birman et al., 1991]

- Only stable messages are delivered
- **Stable message**: a message received by all processes in the message's group view
- Assumptions (can be ensured by using TCP):
  - Point-to-point communication is reliable
  - Point-to-point communication ensures FIFO-ordering

29

Virtual Synchrony Implementation: Example

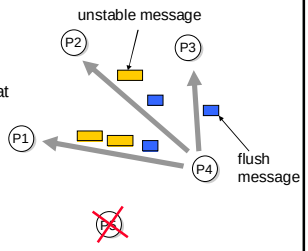
- G_i = {P1, P2, P3, P4, P5}
- P5 fails
- P1 detects that P5 has failed
- P1 send a "view change" message to every process in G_{i+1} = {P1, P2, P3, P4}



30

Virtual Synchrony Implementation: Example

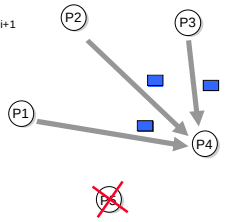
- Every process
  - Send each unstable message m from G_i to members in G_{i+1}
  - Marks *m* as being stable
  - Send a flush message to mark that all unstable messages have been sent



31

Virtual Synchrony Implementation: Example

- Every process
  - After receiving a flush message from any process in G_{i+1} installs G_{i+1}



32

Message Ordering

- **FIFO-order**: messages from the same process are delivered in the same order they were sent
- **Causal-order**: potential causality between different messages is preserved
- **Total-order**: all processes receive messages in the same order
- Total ordering does not imply causality or FIFO!
- Atomicity is orthogonal to ordering

33

Message Ordering and Atomicity

Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

34