CS 731: Blockchain Technology And Applications

Sandeep K. Shukla

IIT Kanpur

C3I Center

Acknowledgement

- X. Yuan, FSU
- Jose Calvo-Villagran, Uwaterloo
- Miguel Castro and Barbara Liskov, MIT

Byzantine General's Problem (Lamport et al.)

- A set of fully connected nodes (processes, processors, threads etc)
- A commander Node which sends a command (Attack or Retreat – 1/0) to all nodes
- Some nodes are malicious and may misreport what they heard
- The commander also may be malicious and send different commands to different nodes
- The nodes that are not malicious must agree on what to do (Attack or Retreat) by agreeing on the value that they think was sent by the commander
- Is it possible to have a distributed algorithm without a central control to do this?

Agreement Problem

- all sites must agree on a value, say, 0 or 1
  - example: decision to commit a DB transaction
- just voting is not enough
- processors may send inconsistent votes to different sites
- Assume:
  - *m* out of *n* processors may fail
  - system is fully connected, pairwise
  - receiver knows sender's identity
  - communications are reliable

Synchronous vs. Asynchronous Distributed System

- synchronous: all processors proceed in ``lock step''
- asynchronous: each processor proceeds at its own pace
- Agreement problem is not solvable in an asynchronous system, even for single-processor failures. (Lynch et al.)
- Failure Modes
  - crash fault
  - omission fault
  - malicious (Byzantine) fault
- Synchronous model allows detection of first two kinds of failures.
- Byzantine failures may be due to hardware or software failures, or due to malicious attacks.

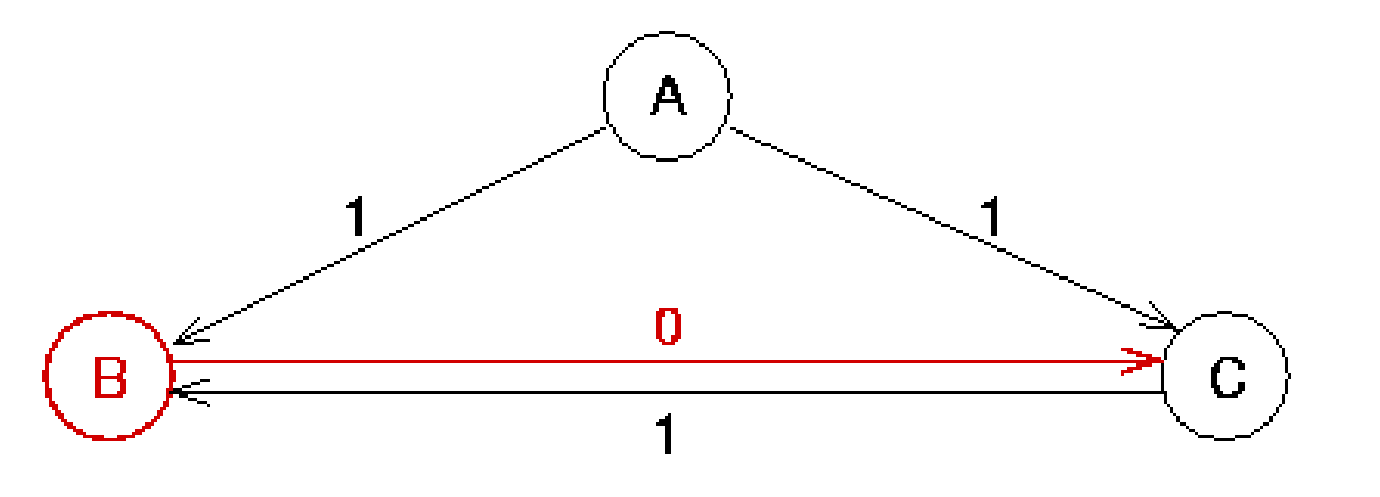
Taxonomy

- All non-faulty processors must agree on value(s) from a non-faulty processor
- Byzantine agreement:
  - The source processor broadcasts its initial value to all other processes.
  - **Agreement**: All nonfaulty processors agree on the same value.
  - **Validity**: If the source processor is nonfaulty, the common agreed upon value by all nonfaulty processors should be the initial value of the source
- consensus:
  - Every processor broadcast the initial value to all other processors.
  - **Agreement**: All nonfaulty processors agree on the same value.
  - **Validity**: If the initial value of every nonfaulty processor is v, then the agreed upon common value by all nonfaulty processors must be v.
- interactive consistency:
  - every processor broadcasts its initial value to all other processors.
  - **Agreement:** All nonfaulty processors agree on the same vector. (v_1, v_2, ..., v_n).
  - **Validity:** If the ith processor is nonfaulty and its initial value is v_i, then the ith value to be agreed on by all nonfaulty processors must be v_i.

Impossibility Results

- Byzantine agreement is impossible if $m > \lfloor (n-1)/3 \rfloor$
  - $e.g.,\ m > \lfloor (3-1)/3 \rfloor = 0$
- Byzantine agreement is impossible with < (m+1) message exchanges

Impossibility of n = 3, m = 1



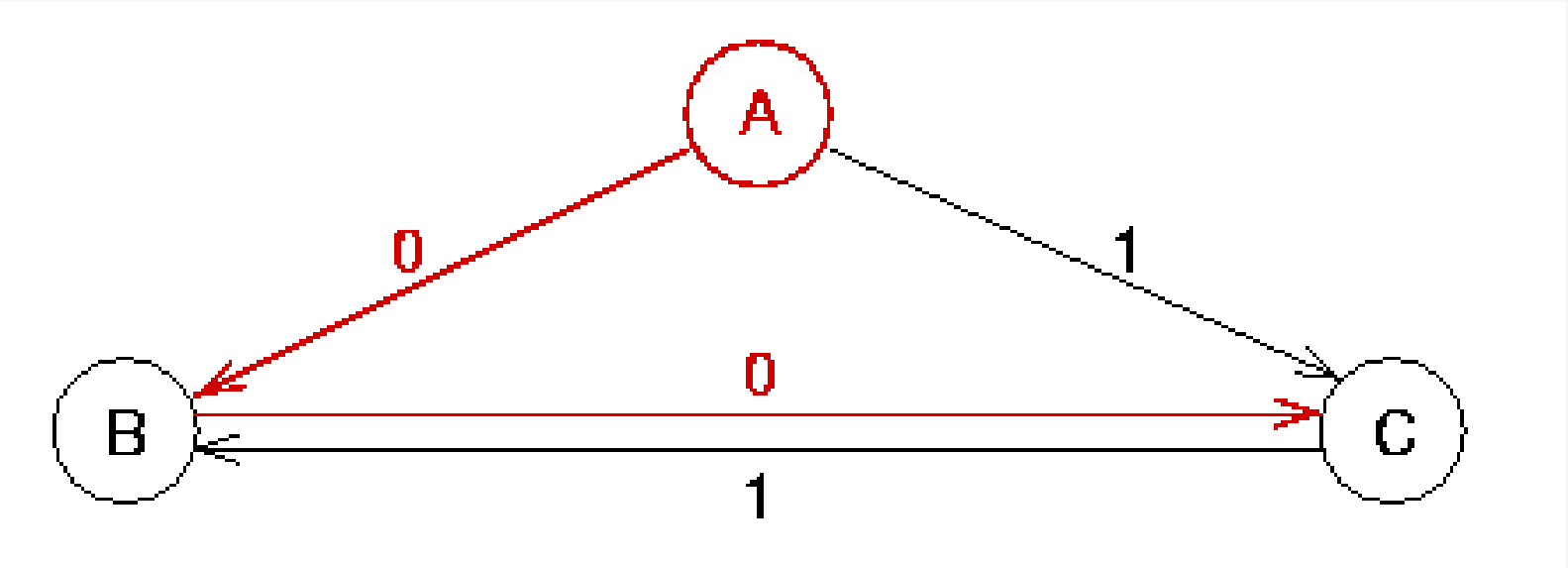
A initiates the agreement protocol and processor B is maliciously faulty.

C sees that B has decided for 0 and A has decided for 1

To satisfy the Byzantine agreement problem, C must decide for 1, since A is not faulty and A has decided for 1.

This implies that the algorithm followed by C (and hence by any non-faulty non-initiating processor) must break ties in favor of the initiating processor.

Impossibility for m = 1, n = 3 (cont.)



When processor A is a traitor, and reports different values to B and C.

B thinks A has decided for 0 and C thinks A has decided for 1.

If the algorithm breaks ties in favor of the initiator, C must decide for 1.

However, B must follow the same algorithm, and so it must decide for 0.

This means we have no agreement among the two nonfaulty processors.

Lamport-Shostak-Pease Algorithm

- solves Byzantine agreement for $n \geq 3m + 1$ processors in the presence of m faulty processors
- recursively defined, as $OM(m, n),\ m \geq 0$
- This is called the ``Oral Message'' algorithm, because the conditions correspond to what we would expect if messages are delivered orally, in person, by pairwise conversations between the parties involved in the consensus.
- Oral Messages:
  - every message that is sent is delivered exactly
  - the receiver of a message knows who sent it
  - the absence of a message can be detected

Lamport's Terminology for Byzantine Agreement Problem

- every processor is a *general*
- the general who initiates the agreement protocol is the *commander*
- the value suggested by the commander is the *order*
- the other generals, to whom the commander sends the order, are his *lieutenants*
- the faulty processors are *traitors*
- the nonfaulty processors are *loyal*

OM(0,S)

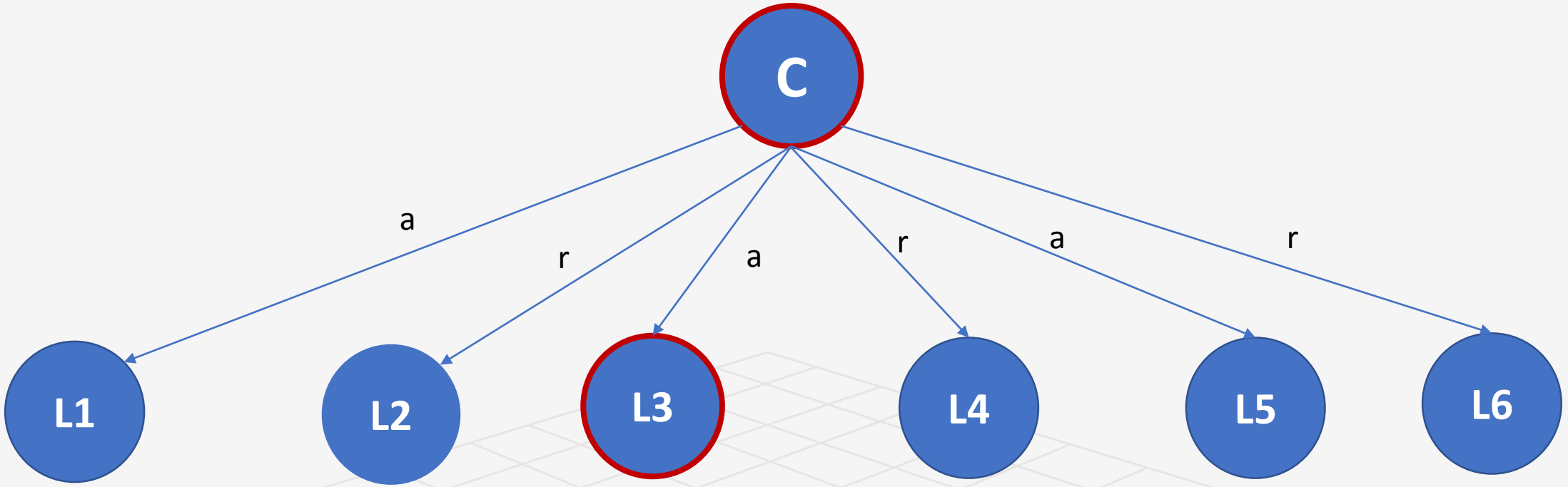
- If there are no traitors, achieving agreement is easy:
  - The commander *j* sends the proposed value *v* to every lieutenant *j* in *S* - {*i*}
  - Each lieutenant *j* accepts the value *v* from *I*

  - *But in reality 0 traitors is not common.*

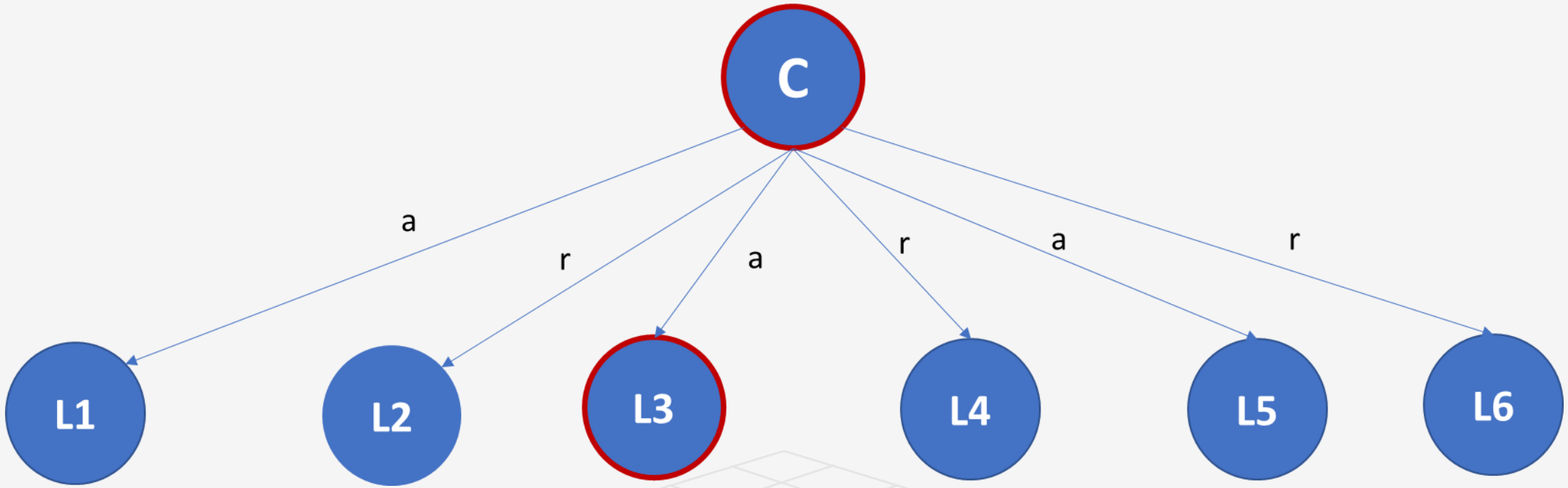
OM(*m*,*S*) for *m* > 0

- ***The commander c sends his value to every lieutenant.***
- ***For each i, v_i is the value Lieutenant i receives from the commander, else RETREAT if he receives no value.***
  - ***Lieutenant i now acts as commander in recursive call $O(m-1, S-\{c\})$ to send the value v_i to each of the $n-2$ other lieutenants.***
- ***For each i such that $i \neq j$, let v_j be the value lieutenant i received from lieutenant j in step (2) (recursive step) or else RETREAT if he received no such value.***
  - ***Lieutenant i uses the value $majority(v_1, v_2, \ldots, v_{n-1})$***

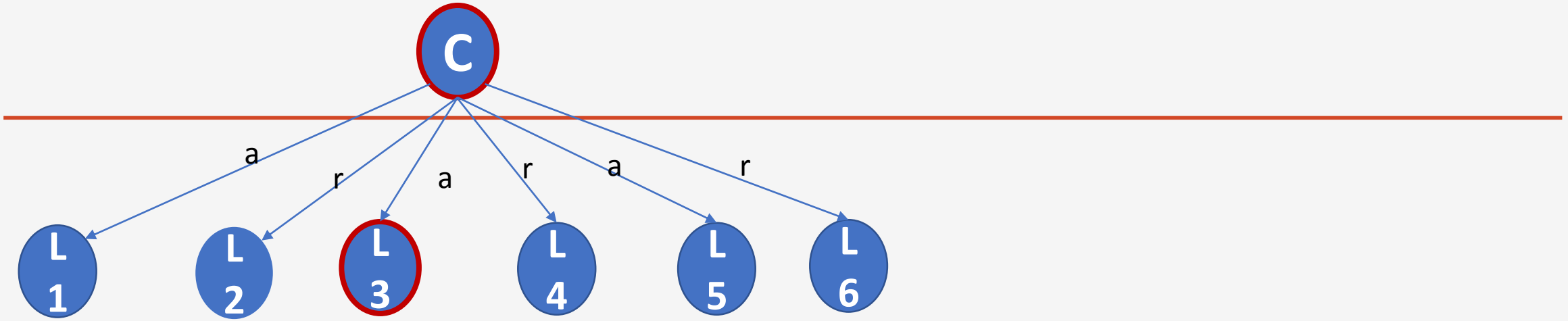
n= 7, m = 2



N = 7, m = 2 (C and L3)



	L1	L2	L3	L4	L5	L6	
C	A	R	A	R	A	R	OM(2, 7)
L3	X	CR	CR	CR	CR	CR	OM(1,6)



	L1	L2	L3	L4	L5	L6	
C	A	R	A	R	A	R	OM(2,7)
L1	X	CA	CA	CA	CA	CA	OM(1,6))
L2	CR	X	CR	CR	CR	CR	OM(1,6)
L3	CR	CR	X	CR	CR	CR	OM(1,6)
L4	CR	CR	CR	X	CR	CR	OM(1,6)
L5	CA	CA	CA	CA	X	CA	OM(1,6)
L6	CR	CR	CR	CR	CR	X	OM(1,6)

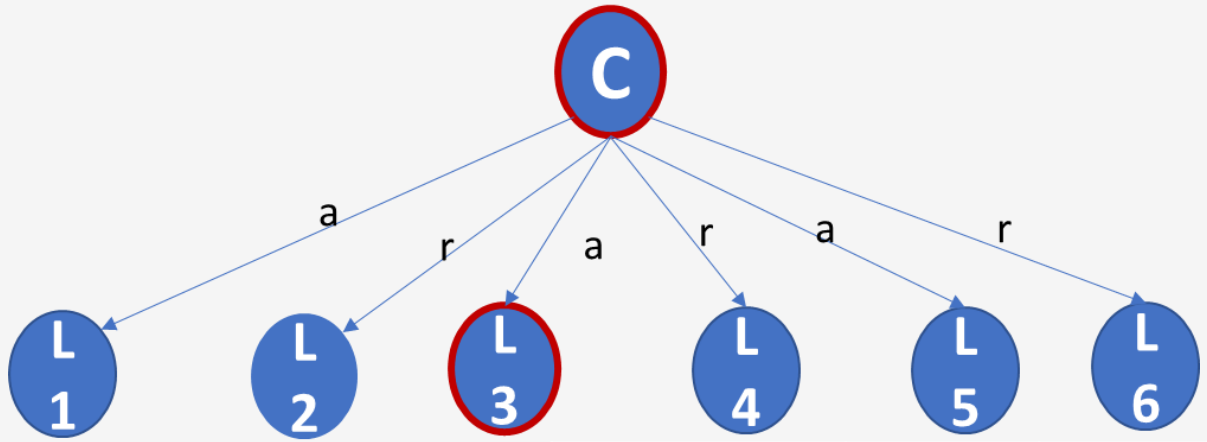
Next step

	L1	L2	L3	L4	L5	L6	
C	A	R	A	R	A	R	OM(2,7)
L1	X	CA	CA	CA	CA	CA	OM(0,5)
L2	X	X	1A	1A	1A	1A	OM(0,5)
L3	X	1R	X	1R	1R	1R	OM(0,5)
L4	X	1A	1A	X	1A	1A	OM(0,5)
L5	X	1A	1A	1A	X	1A	OM(0,5)
L6	X	1A	1A	1A	1A	X	OM(0,5)
Majority(L1)	X	A	?	A	A	A	

Next round

	L1	L2	L3	L4	L5	L6	
C	A	R	A	R	A	R	
L1	X	X	2R	2R	2R	2R	OM(0,5)
L2	C	CA	CA	CA	CA	CA	OM(1,6)
L3	2A	X	X	2A	2A	2A	OM(0,5))
L4	2R	X	2R	X	2R	2R	OM(0,5)
L5	2R	X	2R	2R	X	2R	OM(0,5)
L6	2R	X	2R	2R	2R	X	OM(0,5)
Majority (L2)	R	X	?	R	R	R	

Final Decision Time



	L1	L2	L3	L4	L5	L6
Majority(L1)	A	A	?	A	A	A
Majority (L2)	R	R	?	R	R	R
Majority (L3)	R	R	?	R	R	R
Majority(L4)	R	R	?	R	R	R
Majority(L5)	A	A	?	A	A	A
Majority(L6)	R	R	?	R	R	R
Majority	R	R	?	R	R	R

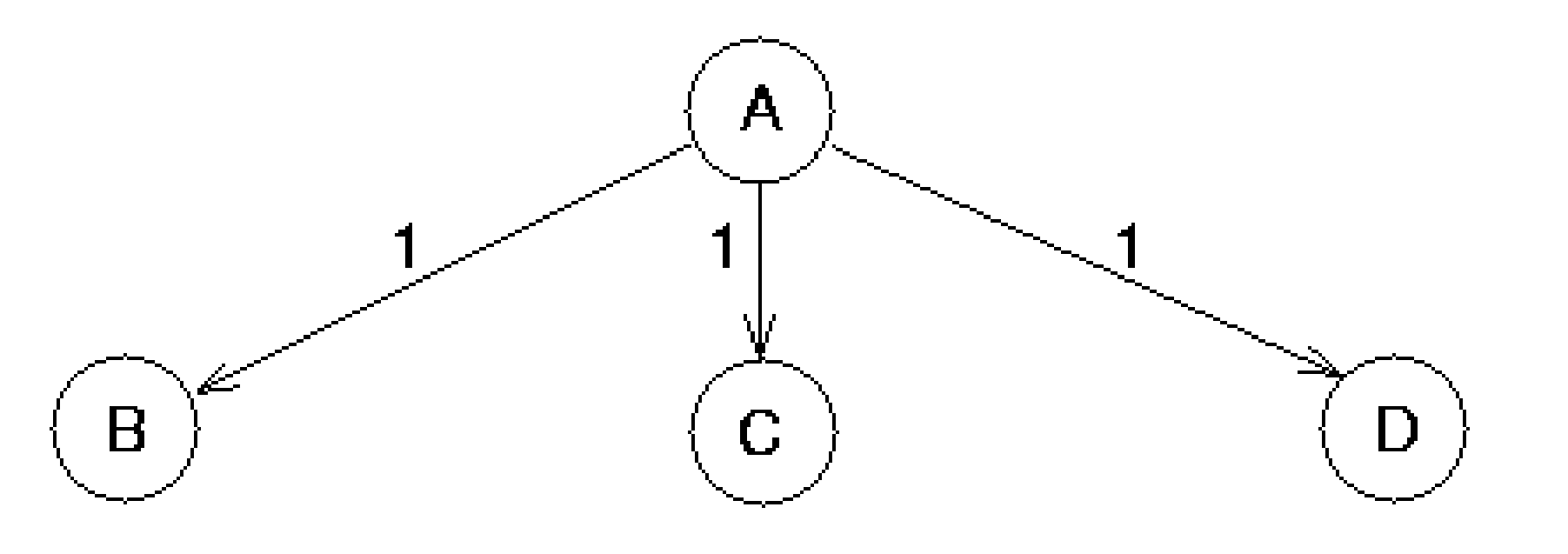
Why does the algorithm recursive?

- Since the messages are transmitted "orally" (not broadcast), the lieutenants must all exchange information about what they received in the previous round, before they can hold the majority vote.
- The ballot would still be easy if we could trust every processor to report accurately what it received.
- However, we must allow fo the possibility that some lieutenants are traitors, and so will report different things to different other lieutenants.
- That is why we need to do a Byzantine agreement on each of the messages that was sent to a lieutenant in the previous round.

Conditions of Byzantine Agreement

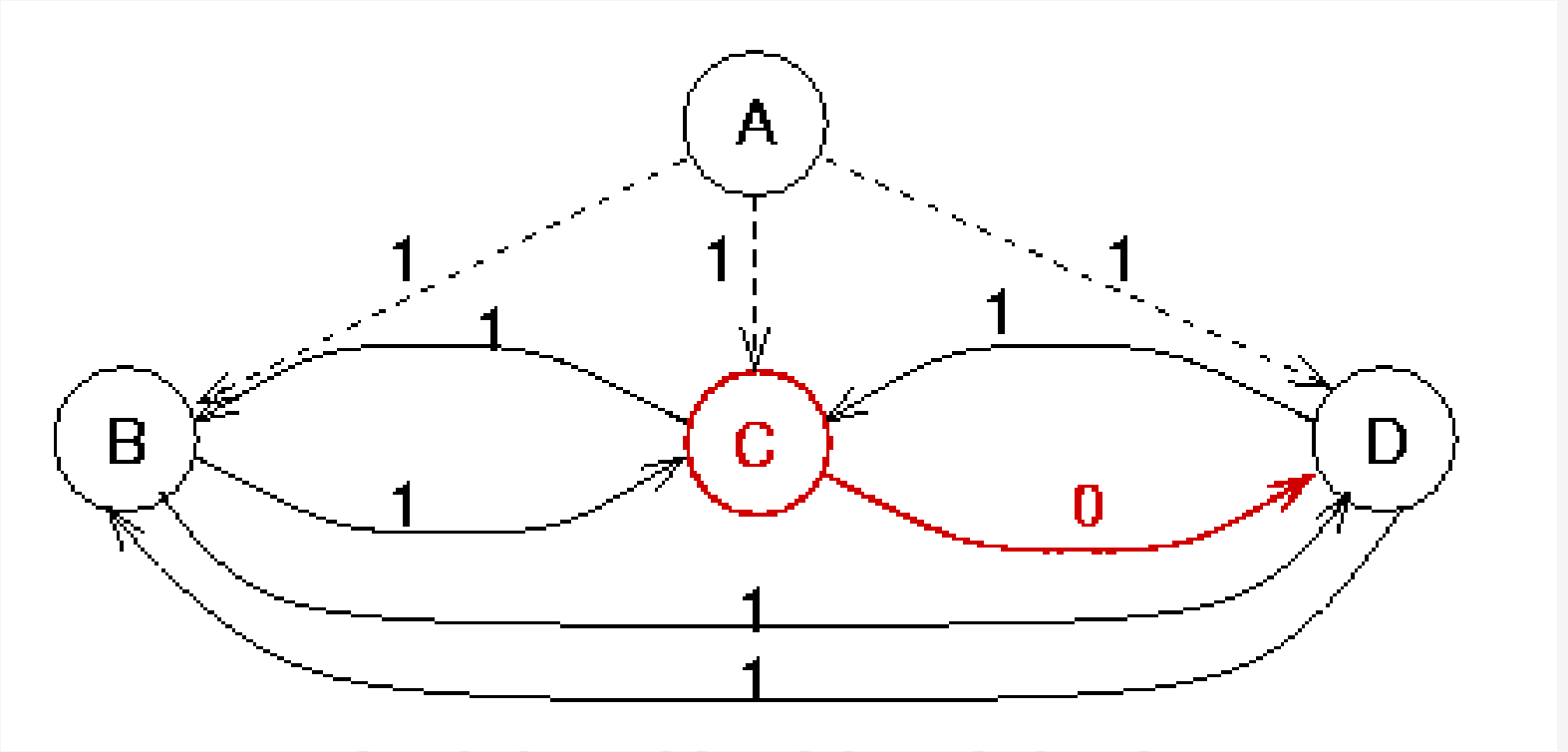
- **Agreement:** All loyal generals agree on the same value.
- **Validity:** If the commander is loyal, then the common agreed upon value for all loyal lieutenants is the initial one given by the commander.

4 nodes example: Commander is Loyal



Round 1: processor A executes OM(1), where processor C is faulty.

Rounder 2 – 4 nodes, m = 1, commander loyal



Round 2: processors B, C, and D execute OM(0). Dashed lines indicate messages sent during the previous round.

Message Complexity

- $T(0, n) = n - 1$
- $T(m, n) = (n - 1)T(m - 1, n - 1), for\ m > 0$
- $T(m, n) = (n - 1)(n - 2)(n - 3) \ldots (n - m - 1) \in O(n^m)$

Problems with Oral Messaging model

- Traitors can lie about what they hear from others
- If we insist that each message is cryptographically signed, and hashed, and cannot be forged, and one has to relay all messages as is because they cannot forge the signature/hash we get signed messages version of Lamport's problem
- This solution does not require $3m + 1$ nodes with m traitors, it works for any $n, m \geq 0$

Algorithm SM(m).

Initially $V_i = \emptyset$.

(1) The commander signs and sends his value to every lieutenant.
(2) For each i:

(A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then

(i) he lets V_i equal $\{v\}$;
(ii) he sends the message $v:0:i$ to every other lieutenant.

(B) If Lieutenant i receives a message of the form $v:0:j_1:\cdots:j_k$ and v is not in the set V_i, then

(i) he adds v to V_i;
(ii) if $k < m$, then he sends the message $v:0:j_1:\cdots:j_k:i$ to every lieutenant other than $j_1, \ldots, j_k$.

(3) For each i: When Lieutenant i will receive no more messages, he obeys the order *choice*(V_i).

Reminder: Byzantine Correctness Conditions

- A commanding general must send an order to his n-1 lieutenant generals such that
  -IC1. All loyal lieutenants obey the same order.
  -IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

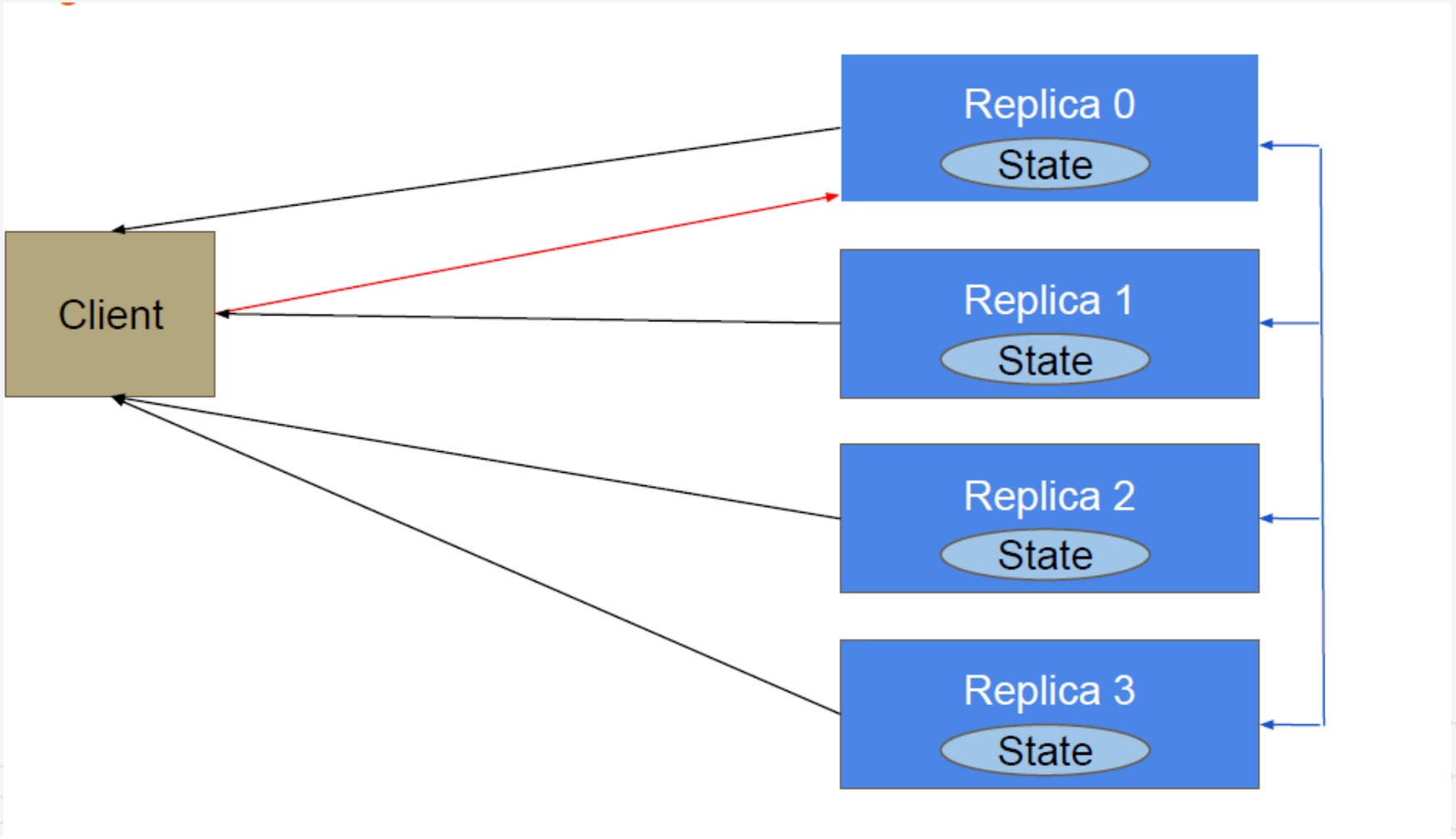
Correctness

- *Theorem 1. For any m, Algorithm OM(m,n) satisfies conditions IC1 and IC2 if there are more than 3m generals and at most m traitors*
- Theorem 2. For any m, Algorithm SM(m) solves the Byzantine Generals Problem if there are at most m traitors
- Both require message paths of length up to m+1 (very expensive)
- Both require that absence of messages must be detected (via time-out (vulnerable to DoS)

Asynchronous System Model

- Asynchronous distributed system where nodes are connected by a network
- Byzantine failure model
  - faulty nodes behave arbitrarily
  - independent node failures
- Cryptographic techniques to prevent spoofing and replays and to detect corrupted messages
- Very strong adversary

System Model



Service Properties

- Any deterministic replicated service with a state and some operations
- Access control to guard against faulty client
- The resiliency (3m+1) of this algorithm is proven to be optimal for an asynchronous system

Basic Setup

- Basic setup:
  - $n = 3m + 1$
- Replicas are identified as $0, 1, \ldots, 3m$
- A view is a configuration of replicas (a primary and backups)
  - Replica p = v mod n is the primary for view v
    - Each replica is deterministic and starts with the same initial state
    - The state of each replica includes the state of the service, a message log of accepted messages, and a view number

The algorithm

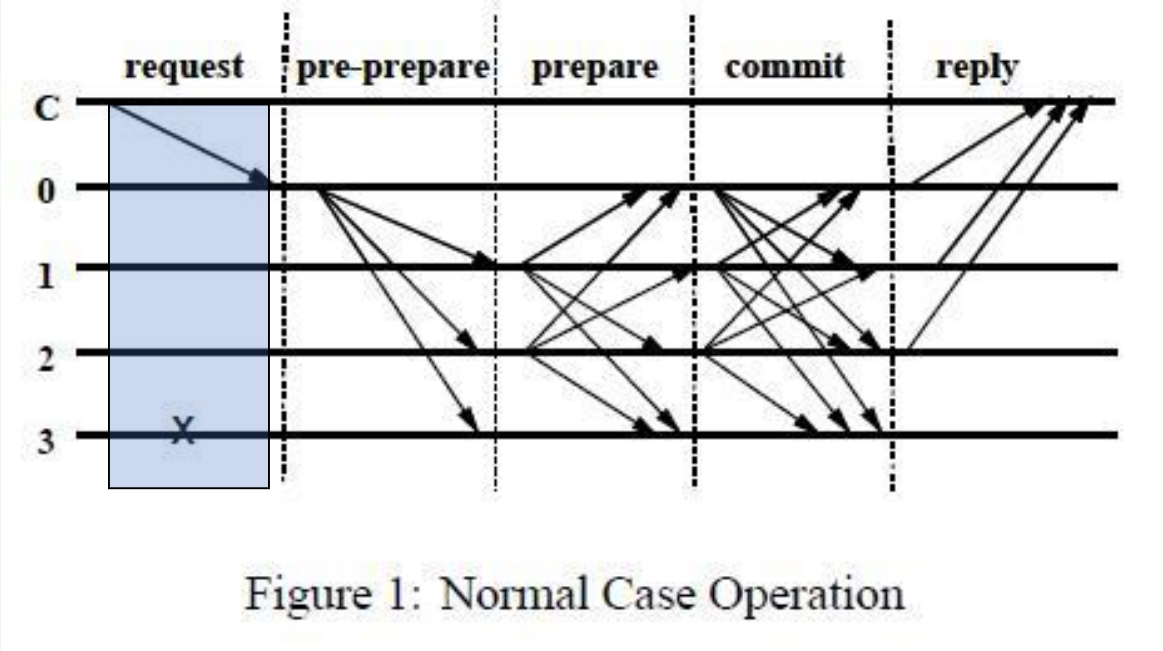


Figure 1: Normal Case Operation

- A client sends a request to invoke a service operation to the primary: $< REQUEST, o, t, c >_{\sigma_c}$
- o = requested operation, t = timestamp, c = client, σ_c = client signature

The Algorithm (2)

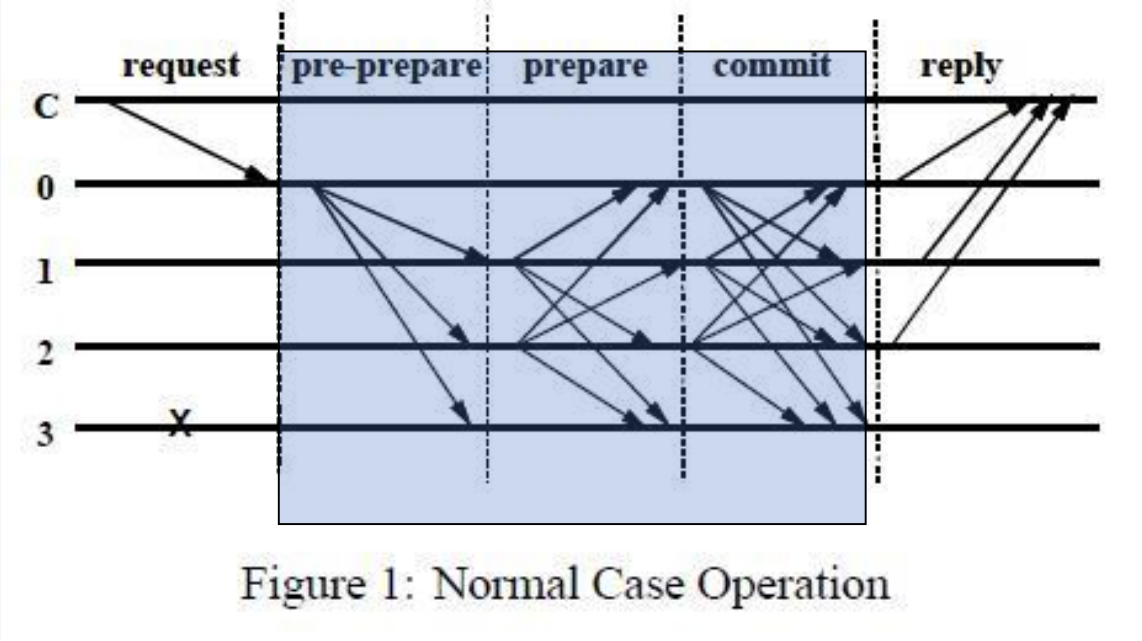


Figure 1: Normal Case Operation

- The primary multicasts the request to the backups (three-phase protocol)

The Algorithm (3)

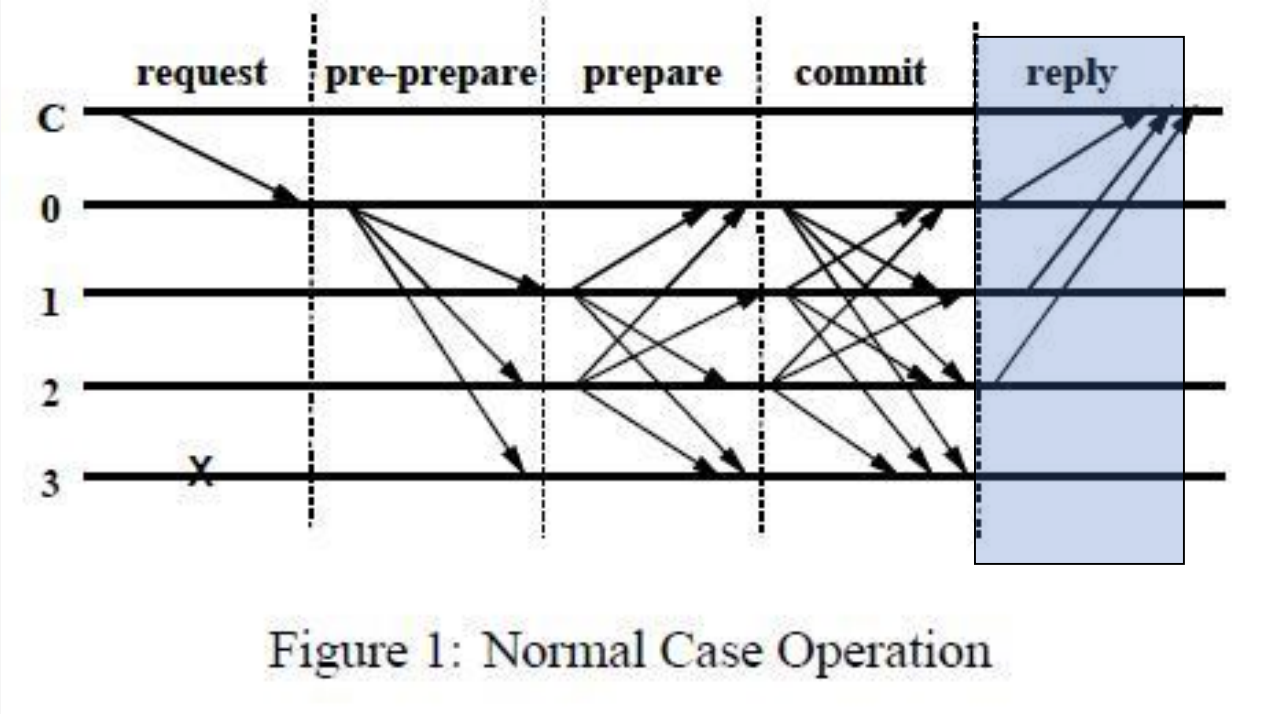


Figure 1: Normal Case Operation

- Replicas execute the request and send a reply to the client $<REPLY, v, t, c, i, r>_{\sigma_i}$
- i = replica, r = result, σ_i = signature of the replica i
- The client waits for m+1 replies from different replicas with the same result; this is the result of the operation

Goals of 3 phase protocol

- Establish a total order of execution of requests (pre-prepare, and prepare)
- Ensure requests are ordered consistently across views (Commit)
- Recall: view is a configuration of replicas with a primary p = v mod n

3 phases

- Pre-prepare
  - Acknowledge a unique sequence number for the request
- Prepare
  - Replicas agree on the sequence number
- Commit
  - Establish total order across the views

Symbol definitions

- Request message o
- Sequence number t
- Signature σ
- View v
- Primary Replica p
- Hash Digest of message d

Pre-prepare

Purpose: acknowledge a unique sequence number for the request

- SEND
  - The primary assigns the request a sequence number and broadcasts this to all replicas
- A backup will ACCEPT the message iff
  - d, v, t, σ are valid
  - (v, t) has not been processed before for another digest (d)

Prepare

Purpose: The replicas agree on this sequence number

After backup i accepts <PRE-PREPARE> message

- SEND
  - multicast a <PREPARE> message acknowledging t, d, i and v
- A replica will ACCEPT the message iff
  - d, v, t, σ are valid

Prepared

Predicate prepared(o,v,t,i) = True iff replica i

- <PRE-PREPARE> for o has been received
- **2m+1**(incl itself) distinct & valid <PREPARE> messages received

Guarantee

- Two **different** messages can never have the same sequence number

i.e., *Non-faulty replicas agree on total order for requests within a view*

Commit

Purpose: Establish total order across views

Once prepared(o,v,t,i) = True for a replica i

- Send
  - multicast <COMMIT> message to all replicas
- All replicas ACCEPT the message iff
- d, v, t, σ are valid

Comitted

Predicate committed(o,v,t,i) = True iff replica i

- prepared(o,v,t,i) = True
- **2m+1**(incl itself) distinct & valid <COMMIT> messages received

Guarantee

Total ordering across views

Executing Requests

Replica i executes request iff

- committed(o,v,t,i) = True
- All requests with lower seq# are already executed

Once executed, the replicas will directly send <REPLY> to the client

- But, what if the primary is faulty? How can we ensure the system will recover?

View Change

- Whenever a lot of non-faulty replicas detect that the primary is faulty, they together begin the *view-change operation*.
  - More specifically, if they are stuck, they will suspect that the primary is faulty
  - The primary is detected to be faulty by using timeout
  - **- Thus this part depends on the synchrony assumption**
  - - They will then change the view
  - - The primary will change from replica p to replica (p+1)mod n