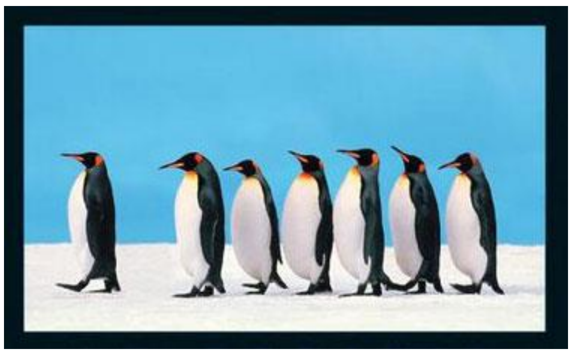
Symmetry breaking problem

Leader Election Algorithms

Master election in GFS?



https://www.pria.com.au/sb_cache/priablog/id/1996/f/leadership.jpg

Requirements of algorithms

Safety: “something bad never happens”

(at most one P_i can enter the elected state)

Liveness: “something good eventually happens”

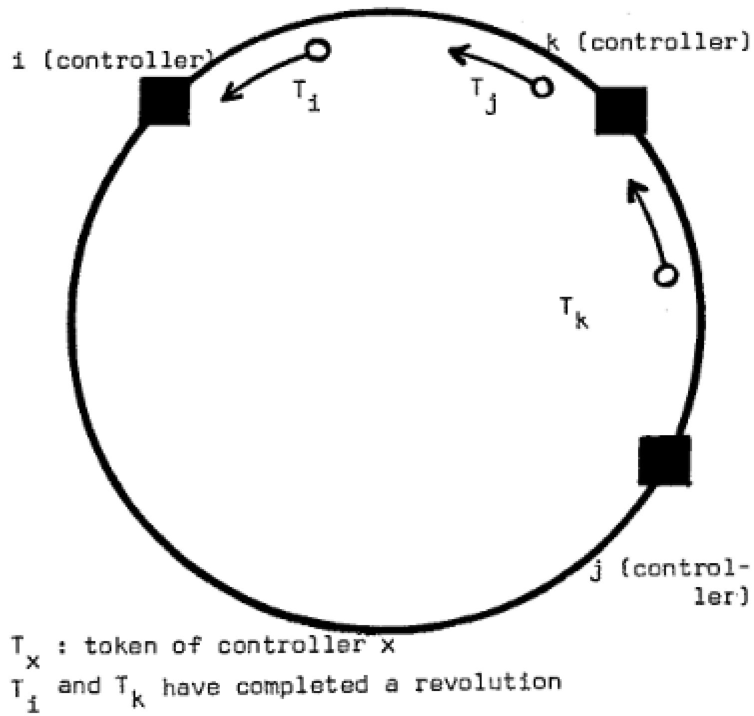
(every P_i will enter either the elected state or the non-elected state)

Distributed Systems - Towards a formal approach

Le Lann, G. (1977).
IFIP Congress Proceedings, pp. 155-160.

Simple algorithm

Applicable to systems organized as a unidirectional ring (nodes send their messages to their right neighbors)



Simple algorithm

No central controller exists (symmetry)

The number n of nodes participating in the election is not known a priori (uniform algorithm)

Each node maintains an active list, consisting of all the id numbers of all active nodes in the system when the algorithm ends

Simple algorithm

If node P_i detects a coordinator failure, it

- creates a new active list that is initially empty,
- sends a message elect(i) to its right neighbor, and
- adds the number i to its active list.

Simple algorithm

If P_i receives a message elect(j) from its left neighbor, it must respond in one of three ways:

- If this is the first elect message it has seen or sent, P_i creates a new active list with the numbers i and j. It then sends the message elect(i), followed by the message elect(j).
- If $i \neq j$, then P_i adds j to its active list and forwards the message to its right neighbor.
- If $i = j$, then P_i receives the message elect(i). The active list for P_i contains all the active nodes in the system. P_i can now determine the new coordinator node.

Simple algorithm – Le Lann (1977)

The algorithm requires $O(n^2)$ messages.

This is clearly so, since each of the n nodes sends a message which is passed to all other nodes.

An improved algorithm for decentralized extrema-finding in circular configurations of processes

Chang, E.G. and Roberts, R. (1979).
Communications of the ACM, Vol. 22, No. 5, pp. 281-283.

Improved algorithm

This note presents an improvement to Le Lann's simple algorithm.

The improved algorithm uses a technique of **selective message extinction** in order to achieve an **average** number of message passes of $O(n \log n)$.

Improved algorithm

When a process P_i receives the identifier k from its right neighbor P_j, it acts as follows:

- i>k: **ignore** the message received from neighbor.
- i<k: forward the identifier k to its left neighbor.
- i=k: due to the assumption on unique identifiers, P_i's identifier must have circulated across the entire ring. Hence P_i can declare itself the leader.

Improved algorithm

(variables)

integer leader

boolean participate false // becomes true when P_i participates in the election

(message types)

PROBE integer // contains a node identifier

SELECTED integer // announcing the result

Improved algorithm

1. When a process wakes up to participate in the election:

- send PROBE(i) to right neighbor;
- participate := true

Text for process P_i

Improved algorithm

2. When a PROBE(k) message arrives from the left neighbor Pj :

- if participate = false execute step 1 first;
- if i > k

    discard the probe

  else if i < k

    forward PROBE(k) to right neighbor

  else if i = k

    leader := i; // declare i is the leader;

    circulate SELECTED(i) to right neighbor

Improved algorithm

3. When a SELECTED(x) message arrives from left neighbor:

- if $x \neq i$

    leader := x;

    forward SELECTED(x) to right neighbor

  else

    discard the message

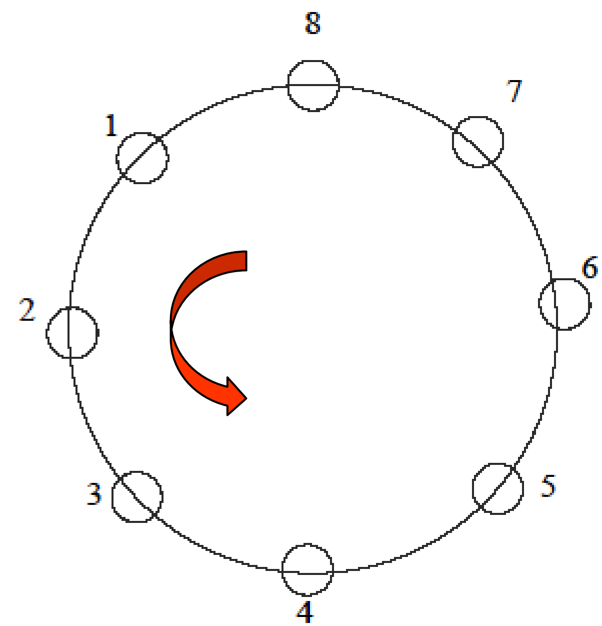
Improved algorithm

Best case

Processes are ordered in the ring in increasing sequence so that message i ($i < n$) only goes once.

Number of message passes
$= (n-1) + n = 2n - 1$

$O(n)$



Improved algorithm

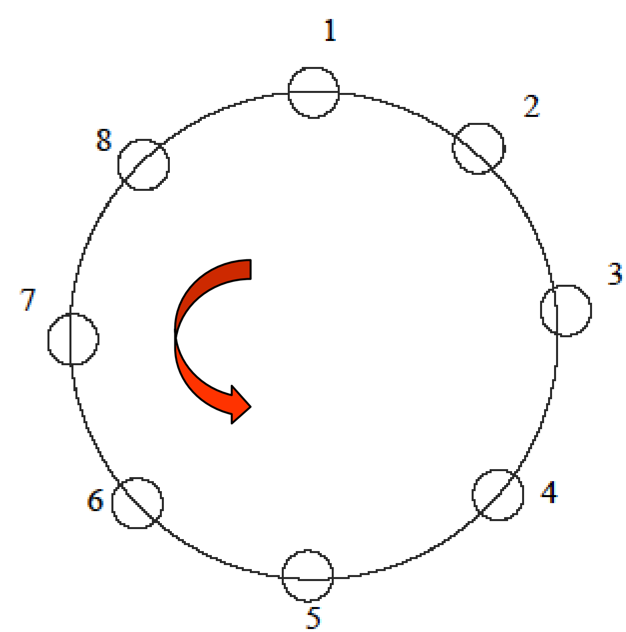
Worst case

Processes are ordered in the ring in decreasing sequence so that message i must be passed i times.

Number of message passes

$$= \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$O(n^2)$



Improved algorithm

Average case (consider all $n!$ rings)

Expected number of message passes

$$= \sum_{i=1}^{n} E(i)$$

$$= 1 + \sum_{i=2}^{n-1} E(i) + n$$

$$= 1 + \sum_{i=2}^{n-1} \sum_{k=1}^{i} kP(i,k) + n$$

Improved algorithm

$P(i,k)$ is the probability that:

A: the $k-1$ counter-clockwise neighbors of i are less than i

AND

B: the kth counter-clockwise neighbor of i is larger than i

$$= P(A \cap B) = P(A)\,P(B/A)$$

$$= \frac{C(i-1,k-1)}{C(n-1,k-1)}\frac{n-i}{n-k} \qquad k \le i$$

Improved algorithm

$$E(2) = \sum_{k=1}^{2} kP(2,k)$$

$$P(2,1) = \frac{C(1,0)}{C(n-1,0)} \frac{n-2}{n-1} = \frac{n-2}{n-1}$$

$$P(2,2) = \frac{C(1,1)}{C(n-1,1)} \frac{n-2}{n-2} = \frac{1}{n-1}$$

$$E(2) = 1\frac{n-2}{n-1} + 2\frac{1}{n-1} = \frac{n}{n-1}$$

Improved algorithm

$$E(n-1) = \sum_{k=1}^{n-1} kP(n-1,k)$$

$$P(n-1,k) = \frac{C(n-2,k-1)}{C(n-1,k-1)}\frac{1}{n-k} = \frac{1}{n-1}$$

$$E(n-1) = \sum_{k=1}^{n-1} k\frac{1}{n-1} = \frac{1}{n-1}\frac{n(n-1)}{2} = \frac{n}{2}$$

$$E(i) = \frac{n}{n-i+1} \qquad 1 \le i \le n$$

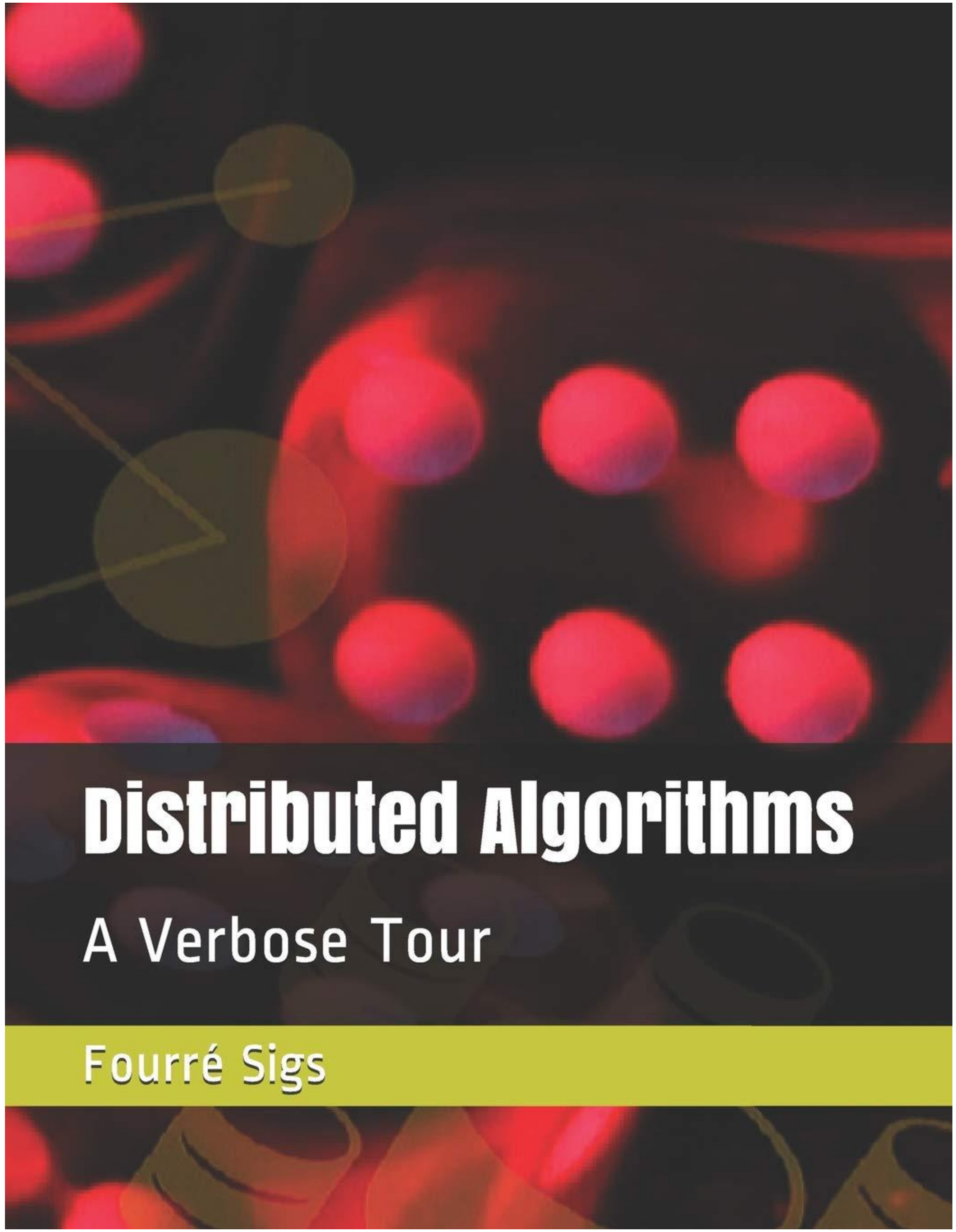
Improved algorithm

$$E(1) = 1$$
$$E(2) = n/(n-1)$$
$$E(3) = n/(n-2)$$
…
$$E(n-1) = n/2$$
$$E(n) = n$$

Expected number of message passes

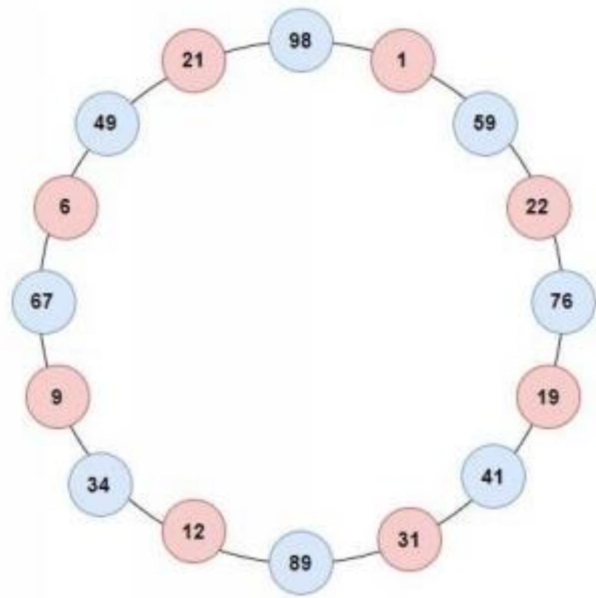
$$= \sum_{i=1}^{n} E(i) = n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$$

$$O(n \log n)$$


Distributed Algorithms
A Verbose Tour
Fourré Sigs

Speeding up the basic algorithm

Hirschberg, D. and Sinclair, J. (1980). *Decentralized extrema-finding in circular configurations of processors. Communications of the ACM*, Vol. 23, No. 11, pp. 627-628.



Bidirectional ring

Speeding up the basic algorithm

This algorithm requires $O(n \, log \, n)$ message passes, in the **worst case**, for finding the largest (or smallest) of a set of n uniquely numbered processes arranged in a circle, in which no central controller exists, and the number of processes is not known a priori.

Speeding up the basic algorithm

In round $k \geq 1$, each active process does:

- Probe circulated to 2^k neighbors on both sides
- P_i is a leader after round k if i is the highest identifier among 2^{k-1} neighbors in each direction

Only a leader after a round proceeds to next round

Speeding up the basic algorithm

```
To run for election:
status := candidate;  maxnum := 1
while status = candidate
        sendboth ("from", myvalue, 0, maxnum);
        await both replies;   // but react to other messages
        if either reply is "no"
                status := lost
        else
                maxnum := 2*maxnum
```

Speeding up the basic algorithm

```
On receiving message ("from", value, num, maxnum):
if value < myvalue    sendecho ("no", value)
if value > myvalue
        status := lost;
        num := num + 1;
        if num < maxnum
                sendpass ("from", value, num, maxnum)
        else
                sendecho ("ok", value)
if value = myvalue    status := won
```

Speeding up the basic algorithm

On receiving message ("no", value) or ("ok", value):

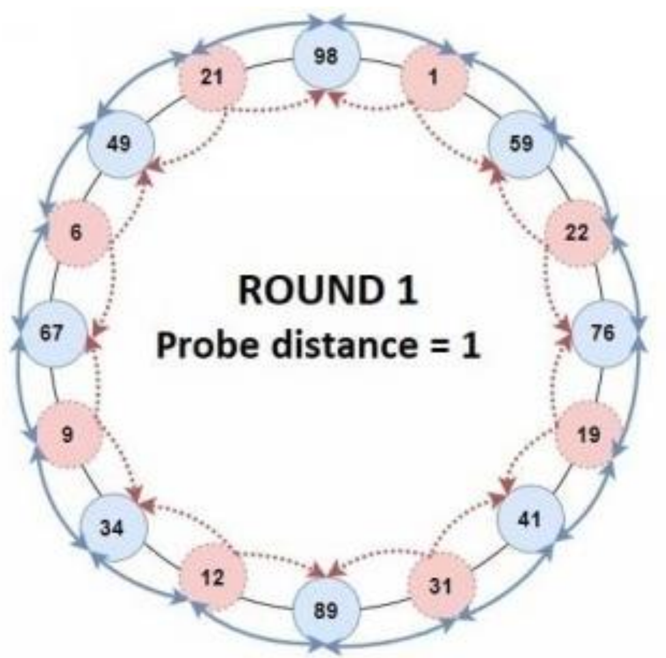
if value ≠ myvalue

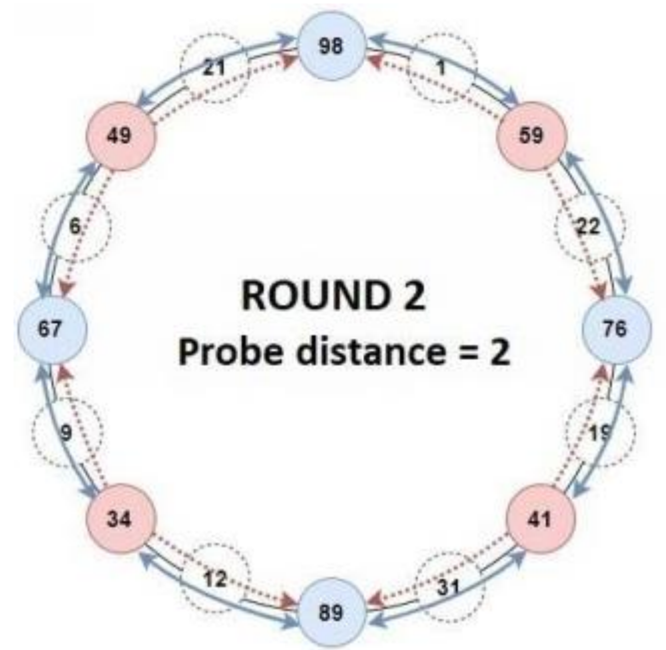
    sendpass the message

else

    this is a reply the process was awaiting

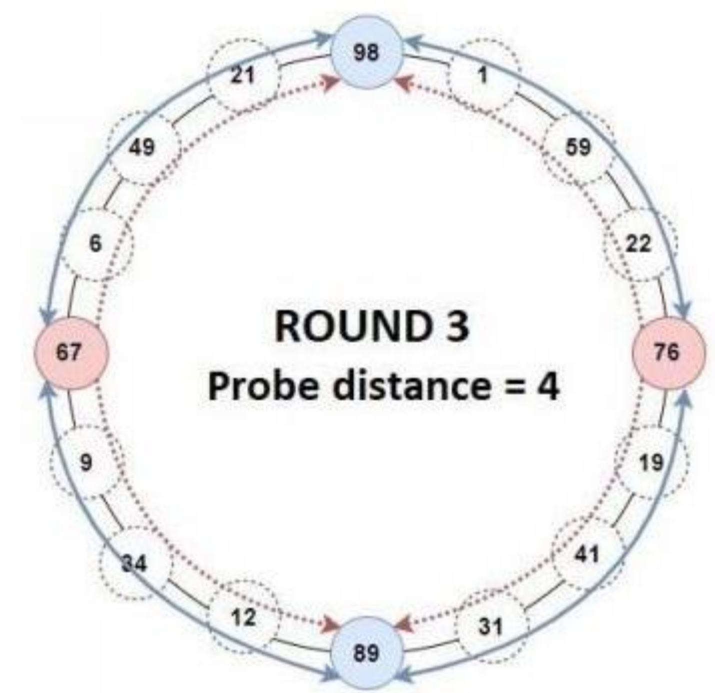
Speeding up the basic algorithm

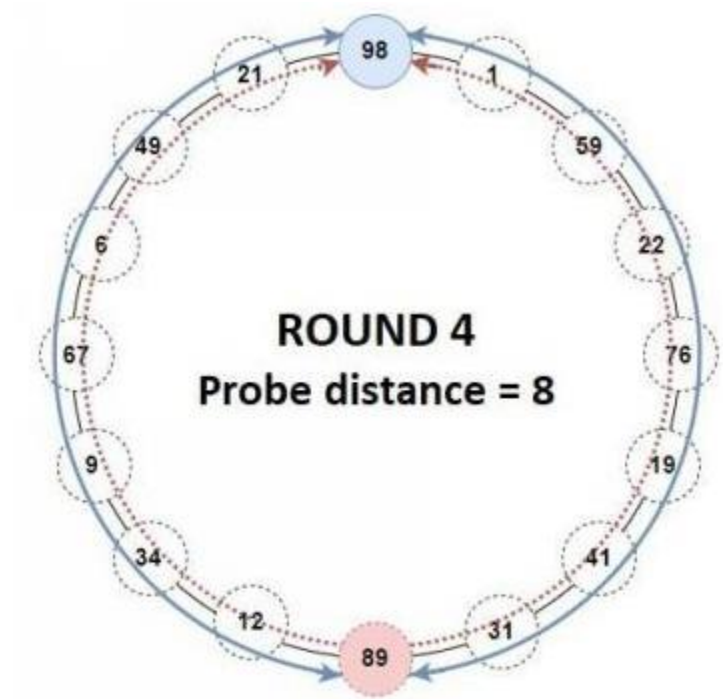




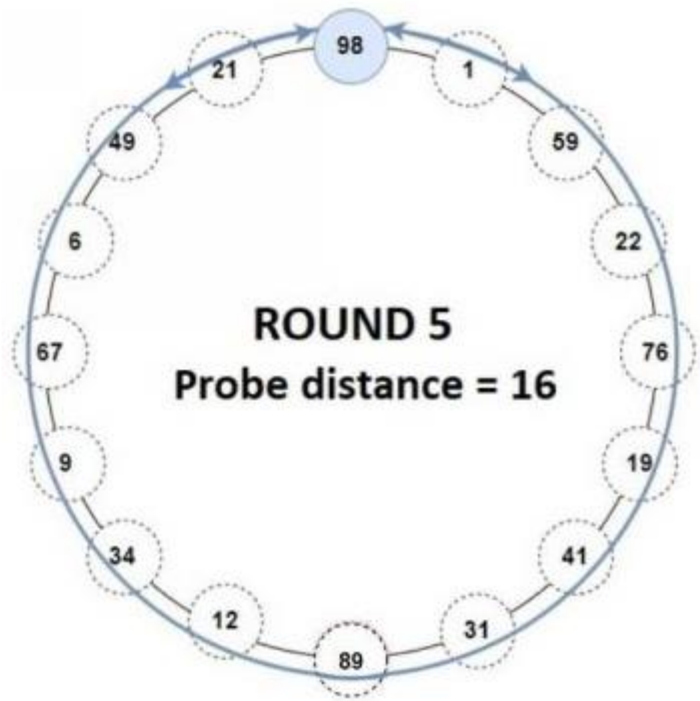
Binary search in both directions on the ring

Speeding up the basic algorithm





Speeding up the basic algorithm



Speeding up the basic algorithm

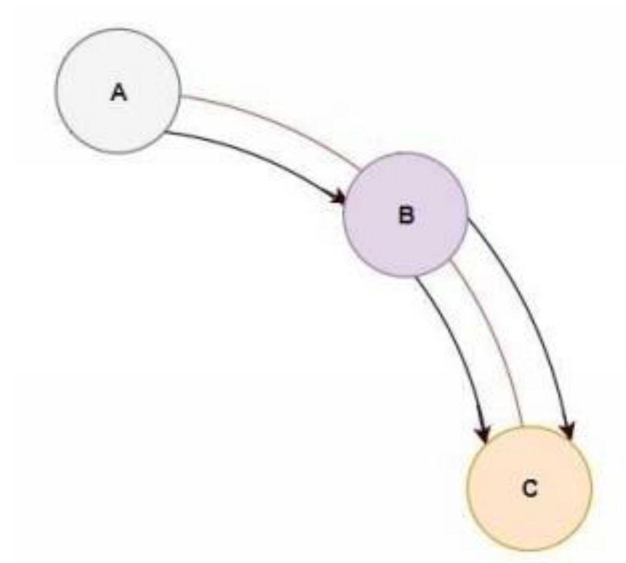
"We conjecture that models in which message passing is unidirectional must, in the worst case, have quadratic behavior and that bidirectional capability is necessary in order to achieve $O(n\,log\,n)$ performance"

Speeding up the basic algorithm

Peterson, G. (1982). An $O(n\ log\ n)$ ***unidirectional*** *distributed algorithm for the circular extrema problem. ACM Transactions on Programming Languages and Systems.* Vol. 4, No. 4, pp. 758-762.

Speeding up the basic algorithm

1. Each process sends its identifier two steps clockwise (so each process sees the identifiers of its neighbor and its next-to-last neighbor).



Speeding up the basic algorithm

2. Each process P makes a decision based on its own identifier and the identifiers it has received. If P's immediate neighbor's identifier is the greatest of the three, then P remains "active" in the algorithm and adopts its neighbor's identifier as its own. Otherwise, P becomes a "relay".
3. Now, each active process sends its identifier to its next two active neighbors, clockwise around the ring. Relay processes simply pass along each message they receive.

Speeding up the basic algorithm

Steps 2 and 3 continue repeatedly, with more and more processes dropping out as "relays" until a process finds that its own immediate neighbor is itself, in which case everyone else has dropped out and it declares itself the leader.

Speeding up the basic algorithm

```
tid := initial value
do forever
        send(tid)
        receive(ntid)
        if ntid = initial value   announce elected
        if tid > ntid
                send(tid)
        else  // tid < ntid
                send(ntid)
```

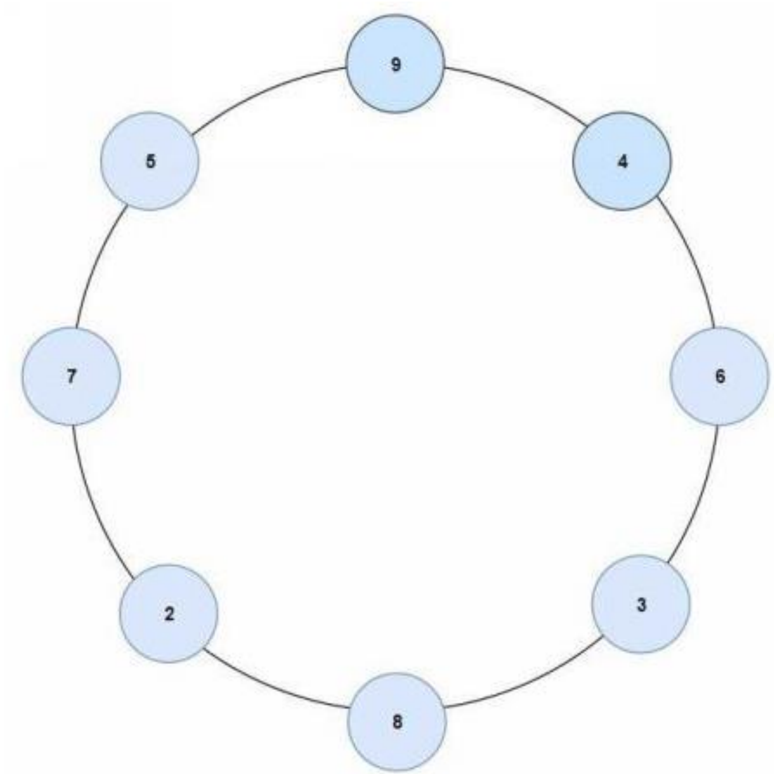
Speeding up the basic algorithm

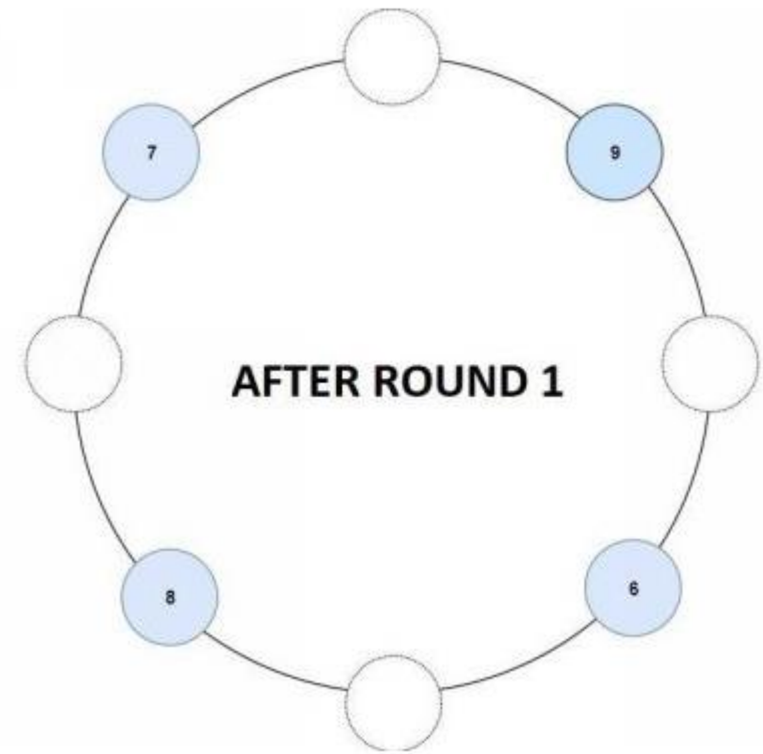
```
        receive(nntid)
        if nntid = initial value   announce elected
        if ntid ≥ max(tid, nntid)
                tid := ntid
        else
                goto relay
end
```

Speeding up the basic algorithm

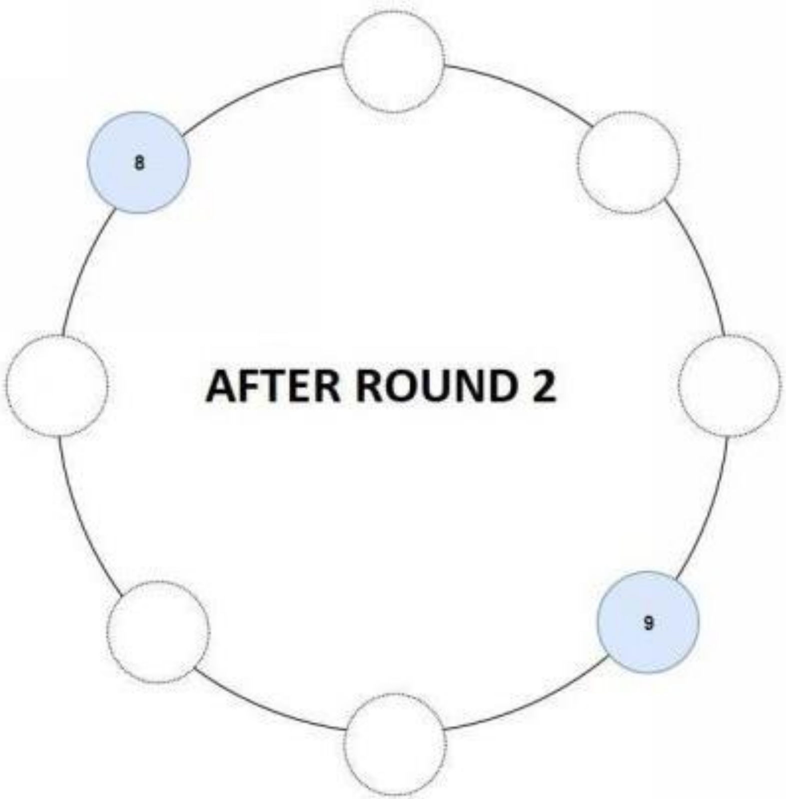
```
relay:
do forever
        receive(tid)
        if tid = initial value   announce elected
        send (tid)
end
```

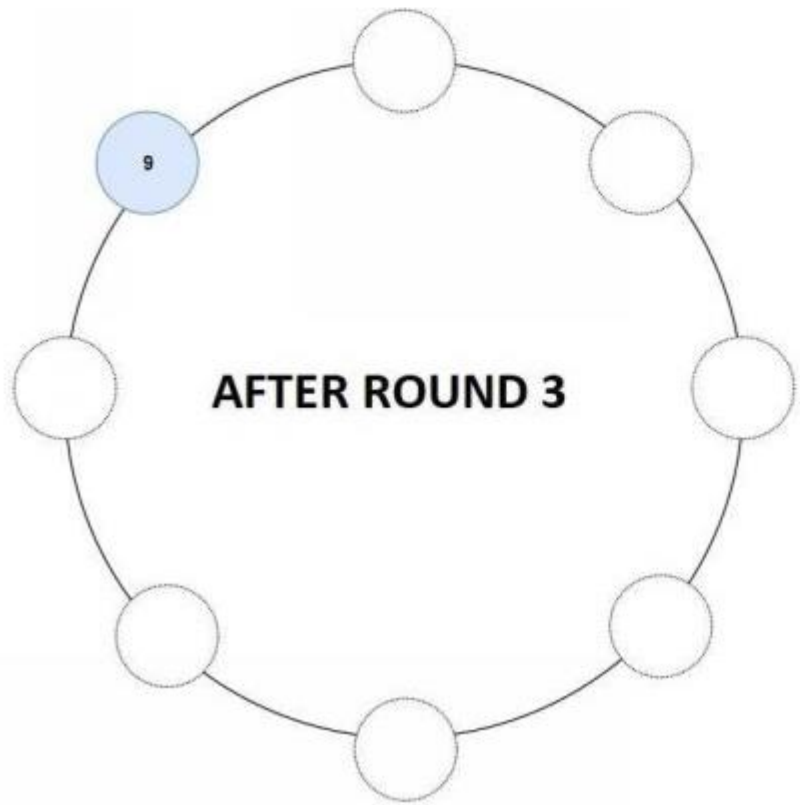
Speeding up the basic algorithm





Speeding up the basic algorithm





Bully algorithm – arbitrary networks

Garcia-Molina, H. (1982). Elections in distributed computer systems. *IEEE Transactions on Computers*, Vol. C-31, No. 1, pp. 48-59.

The Bully Algorithm

Bully algorithm – arbitrary networks

Leader election in synchronous distributed systems where the nodes are known to fail (and possibly recover).

Applicable to systems where every node can send a message to every other node in the system.

Bully algorithm – arbitrary networks

The state vector of node i, S(i), is a collection of safe storage cells which contain data which is crucial for the election and application algorithms.

- S(i).s - Status of node i

  Down, Election, Reorganization, Normal
- S(i).c - Coordinator according to node i
- S(i).d - Definition of the task being performed

Bully algorithm – arbitrary networks

Assertion 1: At any instant in time, for any two nodes i and j in the distributed system, the following must hold.

a) if (S(i).s = Reorganization or S(i).s = Normal) and (S(j).s = Reorganization or S(j).s = Normal) then S(i).c = S(j).c

b) if S(i).s = Normal and S(j).s = Normal then S(i).d = S(j).d

Bully algorithm – arbitrary networks

Assertion 2: If no failures occur during the election, the election protocol will eventually transform a system in any state to a state where:

a) there is a node i with S(i).s = Normal and S(i).c = i

b) all other nodes j which are not failed have S(j).s= Normal and S(j).c = i

Bully algorithm – arbitrary networks

If node P_i sends a request that is not answered by the coordinator within a time interval T, assume that the coordinator has failed; P_i tries to elect itself as the new coordinator.

P_i sends an election message to every node with a higher id, P_i then waits for any of these nodes to answer within T.

Bully algorithm – arbitrary networks

If no response within T, P_i

- assume that all nodes with ids greater than i have failed,
- elects itself the new coordinator,
- sends a message to all nodes with lower ids to inform them about P_i being the new coordinator.

Bully algorithm – arbitrary networks

If answer is received, P_i begins time interval $T´$, waiting to receive a message that a node with a higher id has been elected.

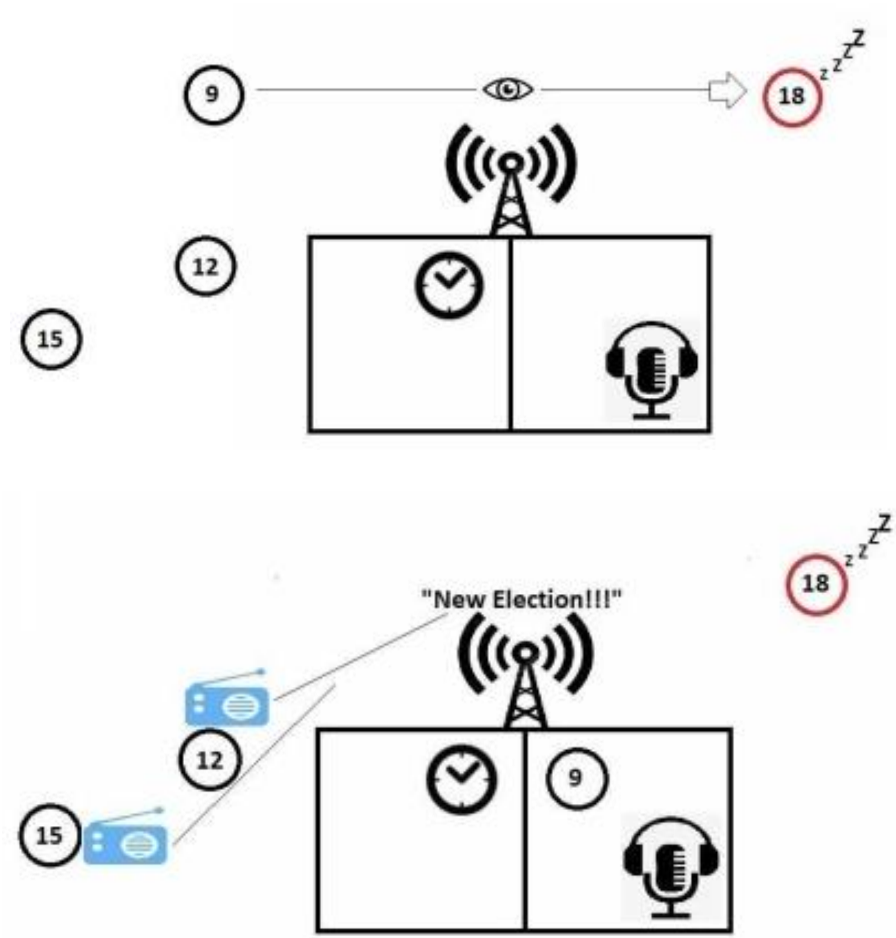
If no message is sent within $T´$, assume the node with a higher id has failed; P_i should restart the algorithm.

Bully algorithm – arbitrary networks

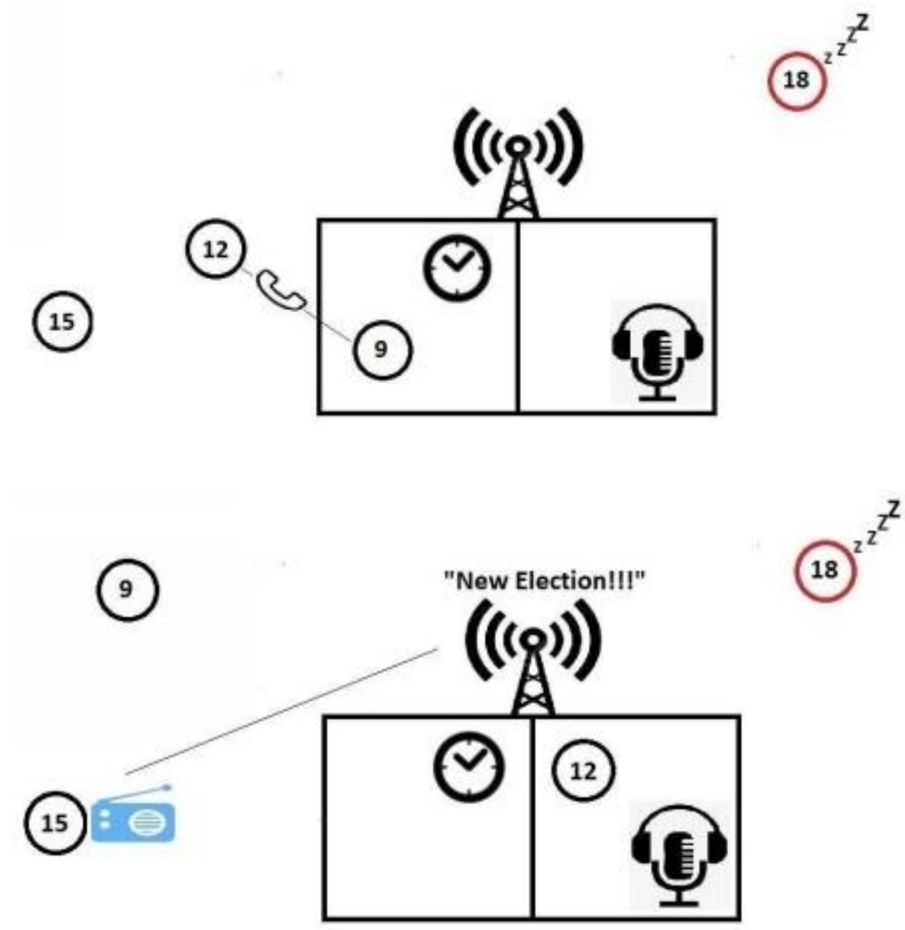
If P_j is not the coordinator, then, at any time during execution, P_j may receive one of the following two messages from node P_i.

- P_i started an election ($i < j$). P_j sends a response to P_i and begins its own election algorithm.
- P_i is the new coordinator ($i > j$). P_j, in turn, records this information.

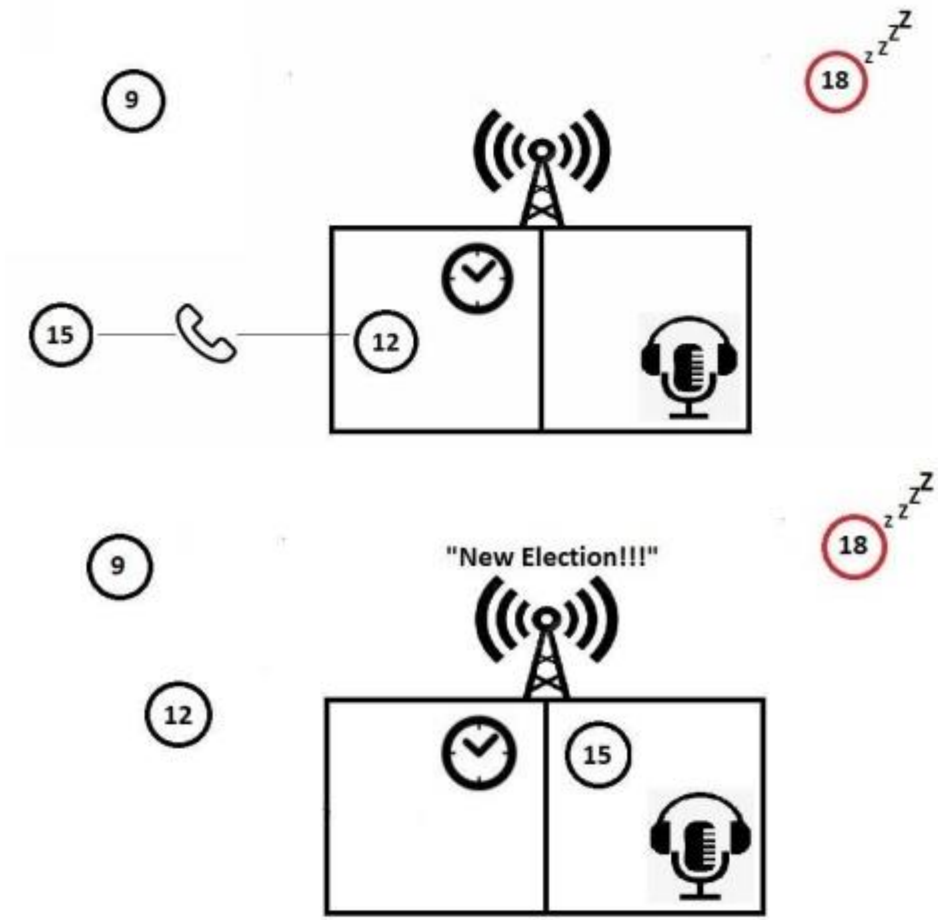
Bully algorithm – arbitrary networks



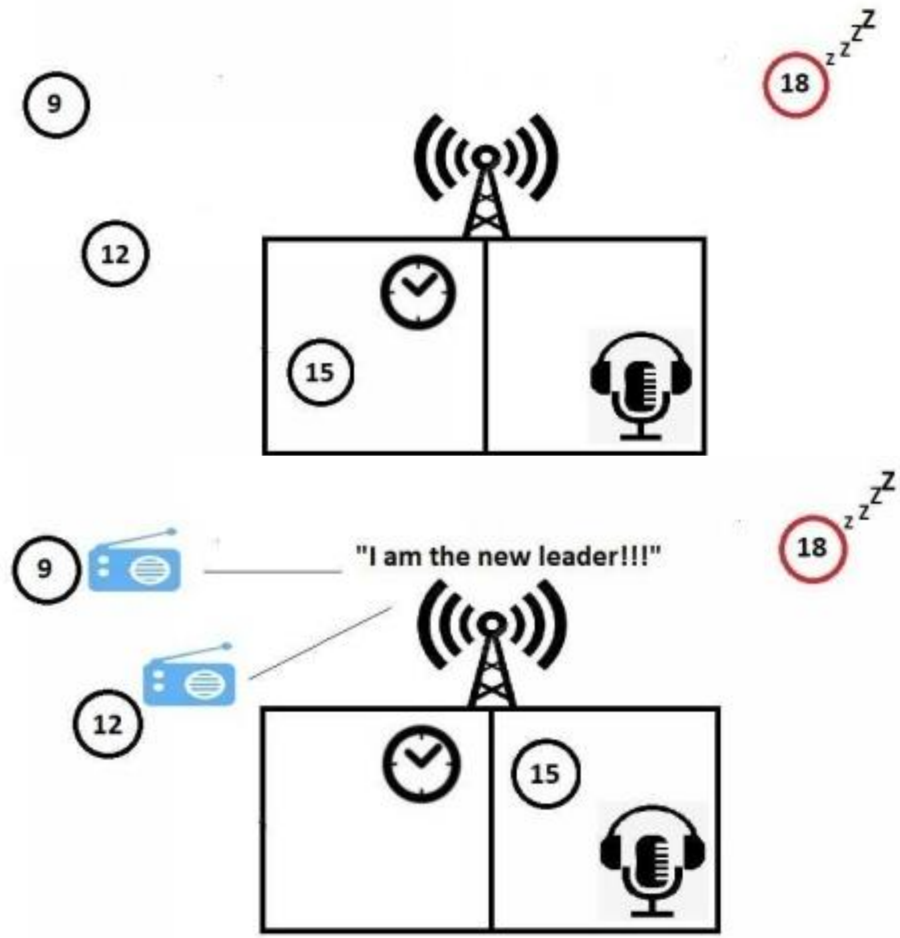
Bully algorithm – arbitrary networks



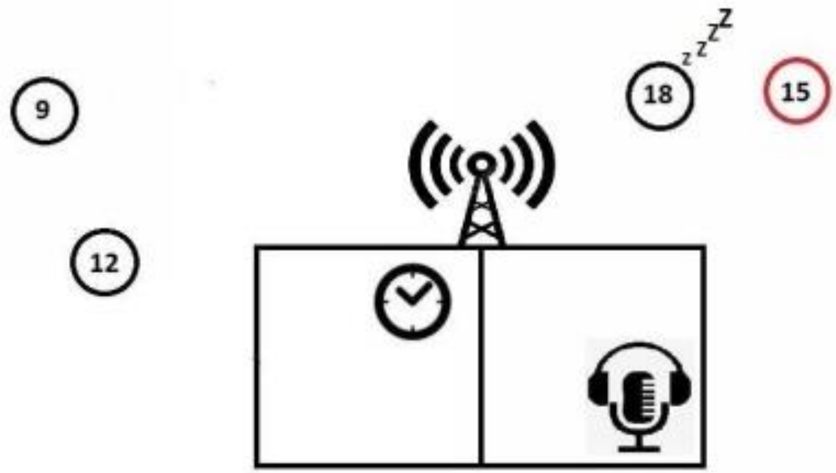
Bully algorithm – arbitrary networks



Bully algorithm – arbitrary networks



Bully algorithm – arbitrary networks



Bully algorithm – arbitrary networks

After a failed node recovers, it immediately begins execution of the same algorithm.

If there are no active nodes with higher ids, the recovered node forces all nodes with lower id to let it become the coordinator node, even if there is a currently active coordinator with a lower number.

Bully algorithm – arbitrary networks