LEADER ELECTION

Abhishek Mathur (20171162)
Kartik Gupta (20171018)
Shaunak Badani (20171004)

The Problem Statement

LEADER ELECTION PROBLEM

In a distributed system we require a leader.

The leader may act as a coordinator for that algorithm, or a load balancer, or might be the only point of contact to the user.

“ A leader is a leader only when he knows he is a leader and everyone else knows he is the leader. There should be no question about his authority. ”

-- Someone (probably lamport or some Indian)

SYSTEM MODEL

1. Each process is either an initiator or a non initiator.
2. No process can fail.
3. The communication is non-fifo and asynchronous (but messages do reach in some finite time).
4. Each process has a unique secret number. This number is not known to any other process in the system, initially. It can only be shared by explicitly passing in messages.
5. Each process is a node in a graph connected by bidirectional edges. This defines the **topology**. Messages can flow only along graph edges.
6. There exists a path between any two nodes of the graph.
7. Any process **z** can be elected as a leader if it satisfies all of the following
    a. All the other nodes in the system know that **z** is the leader.
    b. **z** itself knows that all the other nodes have chosen it as their leader.
8. The algorithm terminates once leader is elected.

SYSTEM MODEL*__ED__*

To correctly simulate our system model we needed to model 3 constraints above what MPI already offers.

1. Some way of giving each process a unique secret number.
    a. Since MPI already allocates a unique rank to each process. WE just made the rank secret.
2. Making sure that communication only happens over the graph edges.
    a. Each node only knows about it's neighbours.
3. The algorithm should not be aware about the rank of any of its neighbours, unless it is explicitly shared in a message.
    a. Handle communication by labeling neighbours from **0 ... num_neighbours-1**. All the communication is done by indexing into a array of neighbour ranks.

To implement this, we created a library called ***topo*** which abstracts away all these details from the user and provides function equivalent to MPI_send, MPI_recv, MPI_Bcast, MPI_Gather and MPI_Reduce, while ensuring that all of the above constraints are taken care of.

ALGORITHMS

Describing a jerk

1. Shout
2. Bully
3. YOYO

SHOUT

ALGORITHM

We try to make a spanning tree.
The root is elected as the leader.

Algorithm ensures that the root
was an initiator.

The initiators initially send out a TEST message to all the other nodes.
All the nodes then wait to receive a message before executing further code

There are 3 other messages: ACKNOWLEDGEMENT,
VICTORY and REJECT

Node Receives TEST

It compares the **root** of the test msg, with its own and takes the **best**.

Best here means the one that is bigger.
Root here means the rank of the root of the tree of the sender.

We can see a TEST message as an invitation to join the tree to which the sender belongs and which is rooted at a process of rank TEST.root

Node Receives TEST

If the receiver deems itself **best**,
It ignores the TEST

If the sender is deemed **best**,
Receiver sets it’s **root** as TEST.root,
It sets its **father** as the **neighbour_idx** of sender,
It sends all it’s neighbours except the father, a TEST

If neither is deemed the BEST,
It sends the sender a REJECT

Node sends ACK

When a node receives ACK or REJECT (corresponding to the current root) from all its children, it sends to its father an ACK.

Root sends VICTORY

When a node receives and ACK or REJECT from all its children and it is the **root of its tree.**

It sends VICTORY,
To all its children.

Node receives VICTORY

It records that the root is the leader,
And sends VICTORY to all its children.

Optimisation

When a node receives a REJECT or it gets a test message with equal root,
It marks that edge as rejected. No messages are sent over rejected edges.

MESSAGE COMPLEXITY

Worst Case

Worst case occurs when for each edge, the REJECT messages are extremely delayed. Hence each time you join a new tree you have to send a message over the each neighbouring edge (from which you haven't gotten a TEST message with equal root).

Hence in this case there are O(E*m) messages. Where E is the number of edges and m is the number of initiators.

In the extreme worst case this algorithm will have message complexity of $O(n^3)$.

Best Case

The test messages from the initiator having the global **best** rank (max rank in the implementation), reach all the nodes before messages from other initiators.

Since we need to send test over every edge and get an ack back,
hence the time complexity is O(E)

TIME COMPLEXITY

Worst Case

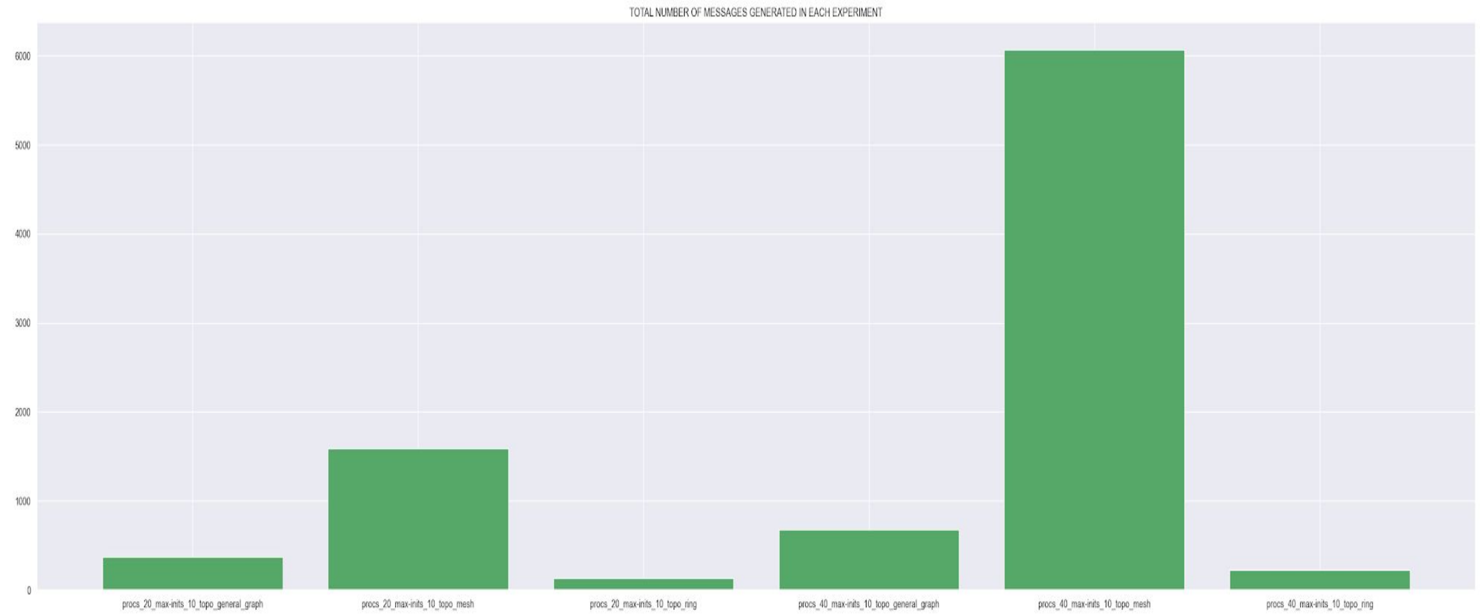
There is delay between when each initiator starts the algorithm and they all start in increasing order of their rank.

Time complexity is O(d*m), where d is the diameter of the graph, and m is the number of initiators.

Best Case

The test messages from the initiator having the global **best** rank (max rank in the implementation), reach all the nodes before messages from other initiators.

Time complexity is O(d) where d is the diameter of the graph.


TOTAL NUMBER OF MESSAGES GENERATED IN EACH EXPERIMENT
6000
5000
4000
3000
2000
1000
0
procs_20_max-inits_10_topo_general_graph
procs_20_max-inits_10_topo_mesh
procs_20_max-inits_10_topo_ring
procs_40_max-inits_10_topo_general_graph
procs_40_max-inits_10_topo_mesh
procs_40_max-inits_10_topo_ring

BULLY

ALGORITHM

Introduction

Bully algorithm is one the famous algorithms for dynamically electing a leader among a set of processes. As the name suggests, the process among all which has the highest ID (in our case the highest rank) is selected as the leader.

Assumptions

- System is synchronous
- Processes may fail any time
- Message delivery system is reliable
- Each process knows its own ID and address
- Failure may arise at any time, including during the execution of the algorithm.

Algorithm

The election comprises of three phases:

- Announcement for election
- Response to an election message
- Victory message on successful election of a leader

The processes which come to know about the system crash or leader crash initiate the announcement phase, and hence are called as the initiator processes. Our implementation of the algorithms deals with one or more than one initiators.

Algorithm

Let’s assume a process P for the sake of convenience in the details.

- When a process P detects system failure, it behaves as the initiator and send announcement message (test message in our case) to all the other connected processes. (Note that there may be multiple such processes at a time)
- When a process P receives test message from a process with higher ID, it replies with a rejection, denoting the incapability of becoming a leader.
- When a process P receives test message from a process with lower ID, it doesn’t reply, and send test message to all other processes in its network.
- In case P receive rejections, from all the neighbouring processes, it means that it has the highest ID among all the processes, and declare itself the leader. (Note that global maximum is achieved in case of a mesh topology, whereas local maximum is also possible in case of other topologies, which is as good as the former)
- Once a process declares itself the leader, it sends victory message to all the processes in the system.

Implementation

- ***Topologies***: We have included several types of topologies like Ring, Mesh and General graph. Where Ring and Mesh are special known cases, General graph generates a network randomly.
- ***Initiators***: As many initiators less than or equal to the number of processes can be declared before the execution. Process with rank 0 is by default an initiator.
- *Roots are defined as similar to the disjoint set heads. The initiators are their own roots and all other processes have roots equal to -1. When the message passing starts, each process updates its root as the source process root.*
- Initiator process starts the algorithm by sending test messages to all the neighbouring nodes.
- Every edge allows only one test message to be sent over it. For example, if P1 has sent a test message to P2, then it can't send it again, however P2 to P1 is still allowed.

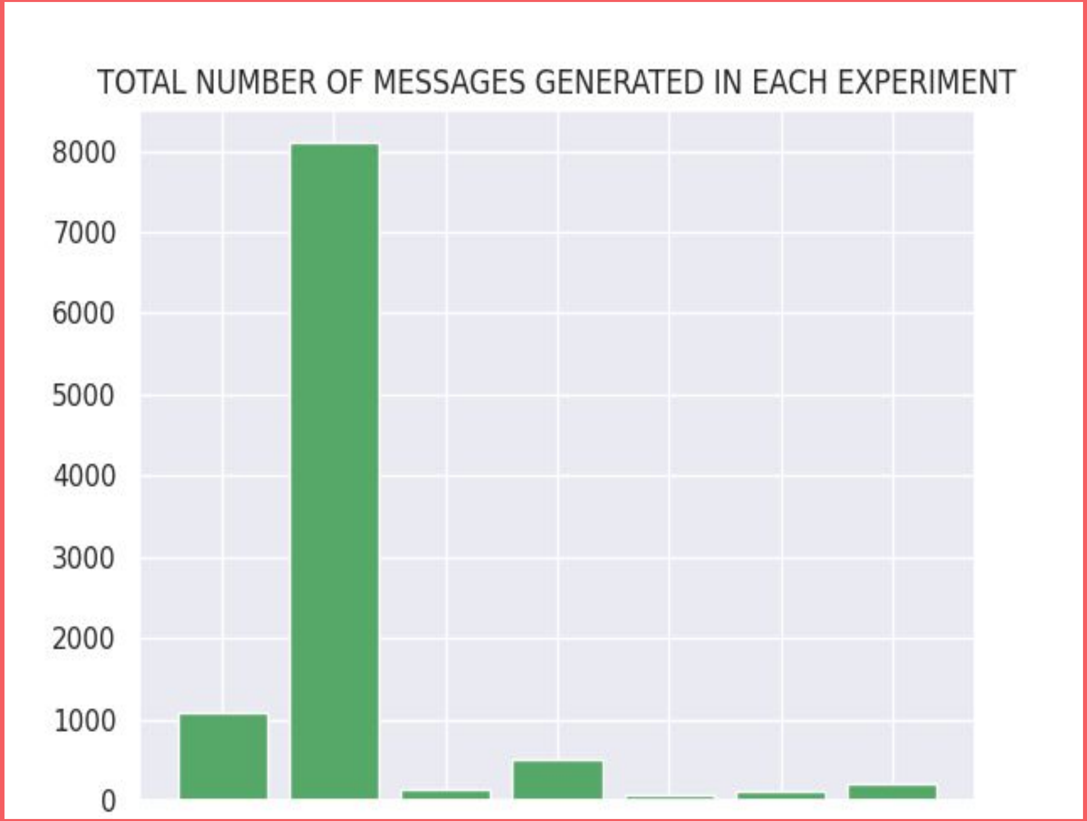
Implementation

- With respect to the above mentioned conditions, the reply messages are sent. That is, in case the current process has lower rank, then the Reject message is sent otherwise none.
- The algorithm continues for each node, till a node receives rejection messages from all its neighbouring nodes, denoting that it’s the highest ranked process (globally or locally, doesn’t matter). Therefore, it announces its root as the leader. A global flag is maintained to avoid appointing multiple leaders.
- *Note: Instead of declaring itself the leader, the highest ranked process declares root as the leader.*
- Once the leader is appointed, the highest ranked process sends the victory message to all the processes in the network. In case the topology is not Mesh, message passing is done process to process.
- As a final outcome, each process announces its rank and the leader.

COMPLEXITY ANALYSIS

- Announcement (started by the initiators) - Each process sends the test message to all the other processes which have greater IDs than itself. This means the least ID process send N-1 messages, and so on. This leads to message complexity of ***O*(*N*^{2})**.
- To make the algorithm more generalised and distributed, we have restricted each process to know ID only of itself, i.e. each process sends the test message to all other neighbouring processes which respond accordingly. Complexity remains the same i.e. ***O*(*N*^{2})**.
- Similarly, we have ***O*(*N*^{2})** message complexity for the Reject message (alive message) which are sent by the smaller ID processes to the greater ID processes.
- Once, the leader is elected, victory message is sent to all the processes, if a processes gets victory message, it exits the algorithm immediately. So the message complexity for this phase is ***O(N)***.

Hence, the total message complexity is O(N^2).

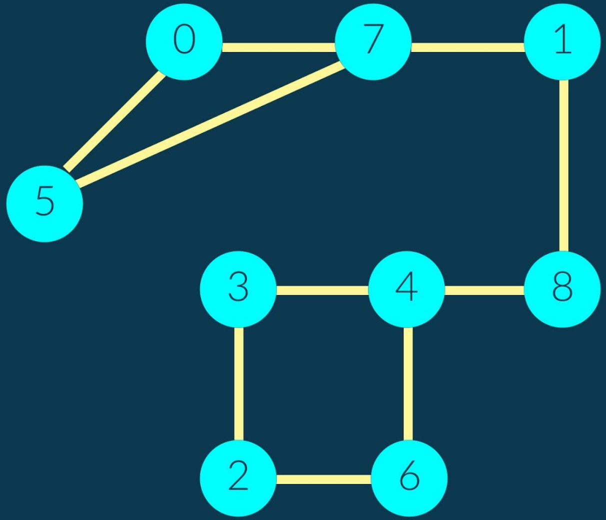


High reaching bars are for mesh topology since the network is very dense. The bottom bars are for ring topology as number of edges is quadratically smaller in comparison with the mesh. And moderate height bars are for randomly generated simple graphs.

YO-YO

ALGORITHM

YO-YO ALGORITHM



The algorithm is divided into 3 parts :

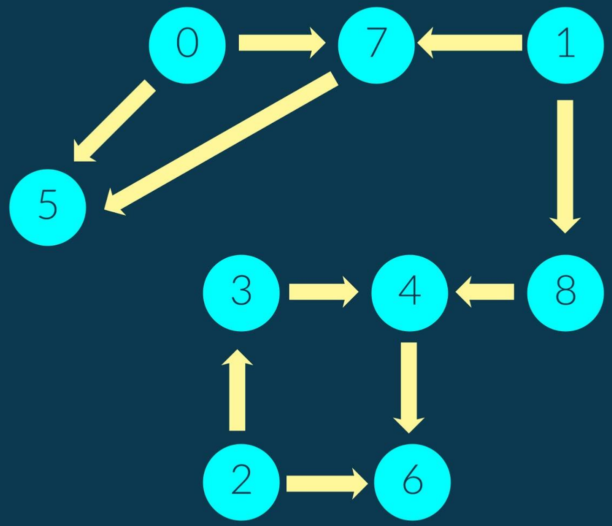
1. Setup phase

The next 2 phases are executed iteratively :

2. Yo- phase

3. -yo phase

SETUP PHASE


$$\begin{tikzcd}
 & 0 \arrow[r] \arrow[dl] & 7 \arrow[dll] & & 1 \arrow[ll] \arrow[d] \\
5 & & & & \\
 & 3 \arrow[r] & 4 \arrow[d] & & 8 \arrow[ll] \\
 & 2 \arrow[u] \arrow[r] & 6 & &
\end{tikzcd}$$

In the setup phase, :

1. Every node sends its value to its neighbours.

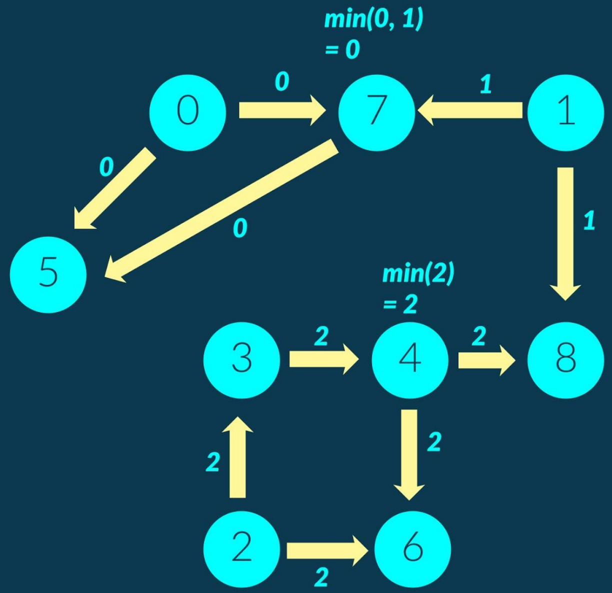
2. Every node receives their neighbours' ranks.

3. If node x receives y , then if

```
 x < y :
 edge x -> y
 else :
 edge y -> x
```

Results in the formation of a Directed Acyclic Graph (DAG).

YO- PHASE



The formation of a DAG leads to 3 types of nodes :

1. Source : Node with no incoming edges [0, 2].
2. Sink : Node with no outgoing edges [8, 6].
3. Internal nodes : nodes which have both incoming and outgoing edges.

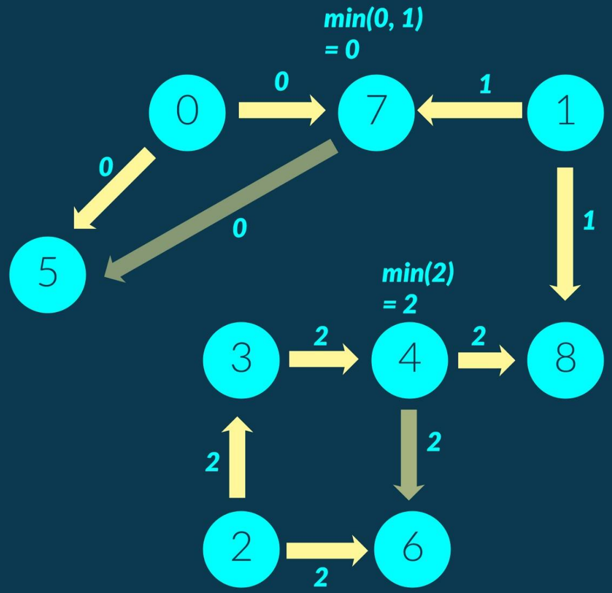
In the Yo-Phase, the 3 nodes do the following :

1. Source : The source sends its value out to all the neighbours.

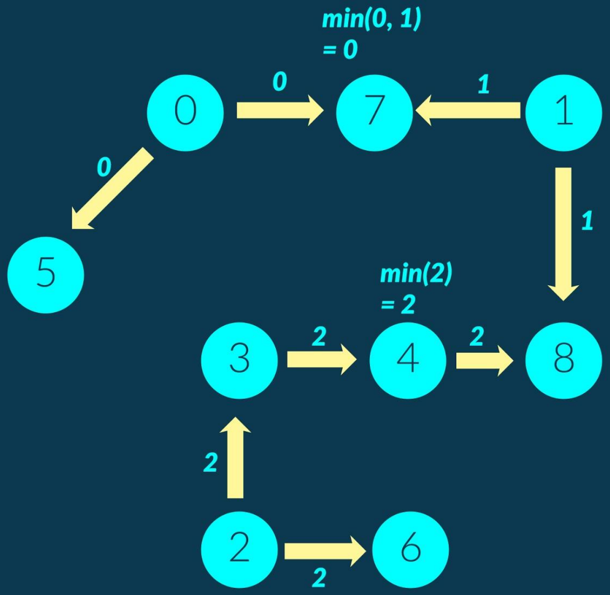
2. Internal nodes : An internal node receives a value from incoming neighbours, computes the minimum of all such values, and forwards this minimum out to all its out-neighbours.

3. Sink : The sink computes the minimum of all such values and proceeds to the next phase.

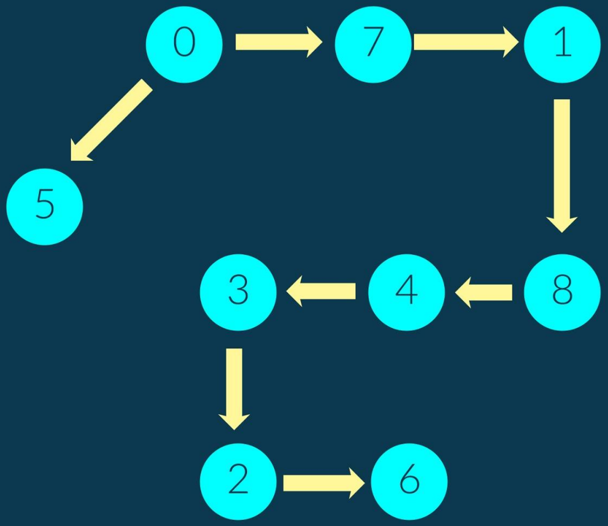
YO- PHASE (PRUNING)



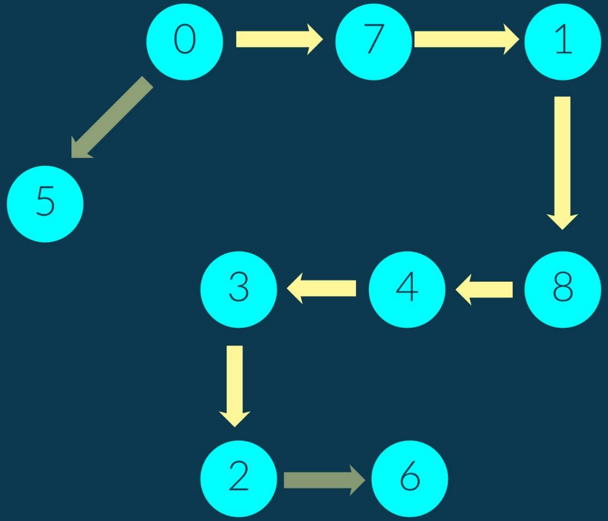
YO- PHASE (PRUNING)



-YO PHASE

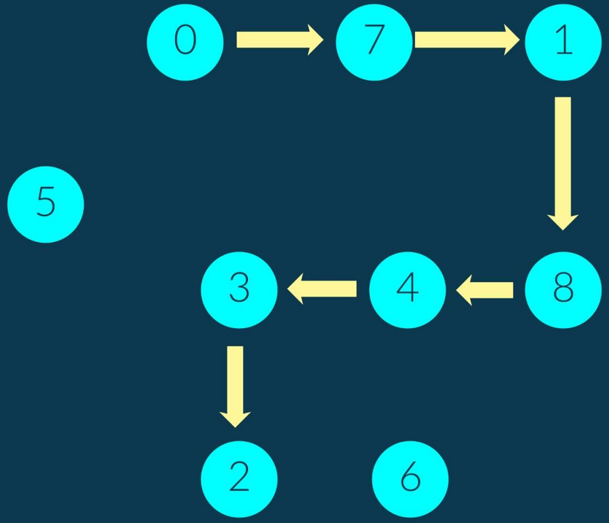


-YO PHASE (PRUNING)



$$\begin{tikzcd}
 & 0 \arrow[r] \arrow[dl] & 7 \arrow[r] & 1 \arrow[d] \\
5 & & & 8 \arrow[l] \\
 & 3 \arrow[d] & 4 \arrow[l] & \\
 & 2 \arrow[r] & 6 &
\end{tikzcd}$$

-YO PHASE (PRUNING)

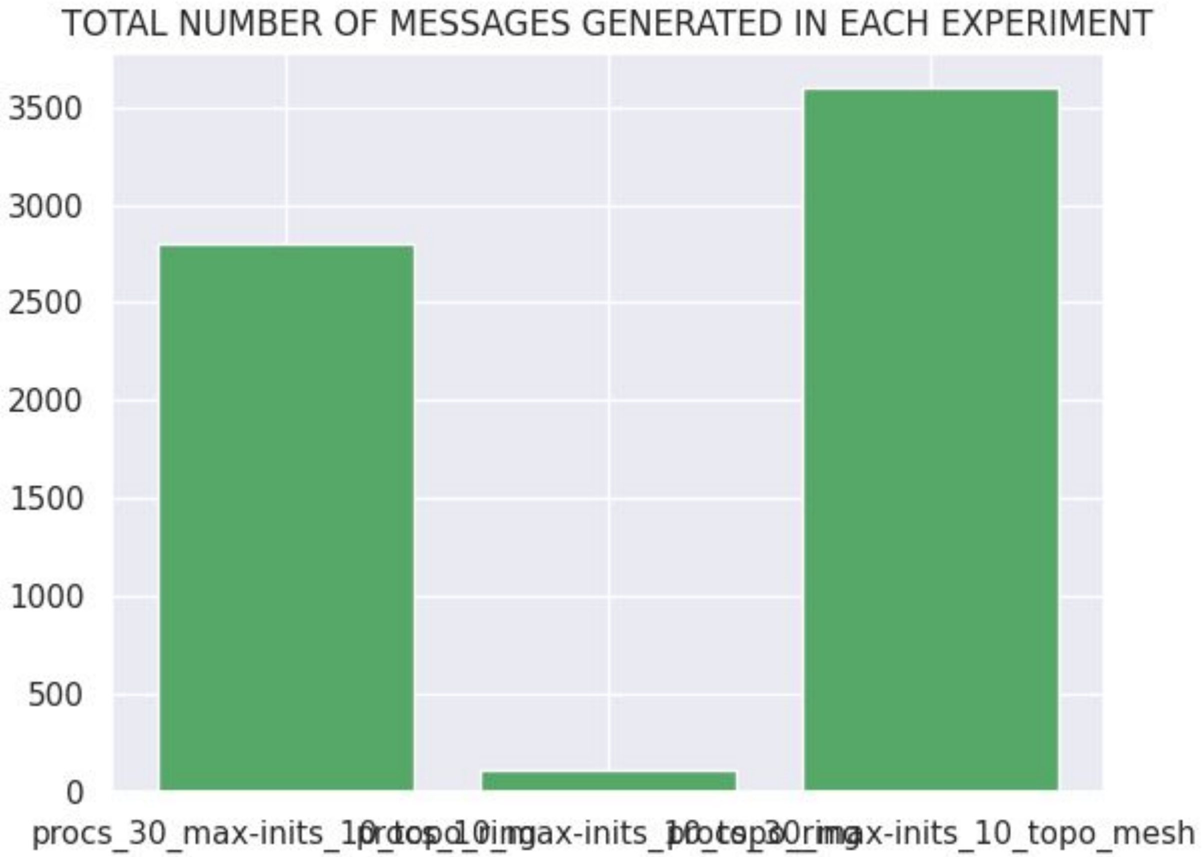


COMPLEXITY

- The setup requires 2 messages to be sent at every edge. Thus the cost of the setup is 2|E|.
- The complexity of the algorithm with pruning is unknown, as the message complexity of pruning is an open research problem.
- However, the complexity of overall algorithm is O(|E| log N).

Complexity contd..

- Let D(1) = G be the original DAG obtained from G as a result of Set-up. Let G(1) be the undirected graph defined as follows: there is a node for each source in D(1), and there is a link between two nodes if and only if the two corresponding sources have a sink in common . Consider now the diameter d(G(1)) of this graph.
- At least one of the two sources will be eliminated by the sink.
- Hence for every 2nd iteration, d(G(2)) < floor(d(G(1)/2).
- Max no of iterations = floor(log(diam(G(1))) + 1
- Since diameter < n,
- Complexity = O(|E|.logN)


TOTAL NUMBER OF MESSAGES GENERATED IN EACH EXPERIMENT
3500
3000
2500
2000
1500
1000
500
0
procs_30_max-inits_10_topo_ring
procs_10_max-inits_10_topo_ring
procs_30_max-inits_10_topo_mesh