Distributed Systems, 2021

Implementation of Leader Election Algorithms in an Unknown Network

Ghazaleh Keshavarz Kalhori[1]

[1]ECE, University of Tehran

June 17, 2022

1 Introduction

In this project we are to implement and test the Echo leader election algorithm with extinction. This is a Las Vegas distributed algorithm, meaning it will end with the probability one but the configuration may be wrong. In the following sections of this report we will further discuss the algorithm, the implementation the result and a brief analysis of the message complexity of this algorithm using the execution results of our own code.

2 Implementation

2.1 The Algorithm

The algorithm starts with a state where one or multiple nodes initiate a leader election algorithm. Initiators start the algorithm by sending a message containing round, id, and sub-tree length. At first, each initiator randomly selects an id within the range of the number of the nodes in the network and sends out the message with round and sub-tree set to zero. At each round a node receiving message acts as below:

The received message is (n', j, s').

If $n' > n$, or $n' = n$ and $j > i$, then the current node makes the sender it's parent and changes to the wave in round n' with the id j. According to echo's wave algorithm, the current node then sends this message to all it's neighbors except the parent.

If $n' < n$, or $n' = n$ and $j < i$, then the current node purges the message.

If $n' = n$ and $j = i$, the current node must wait for all the neighbors to return the message. If all neighbors have returned the message then it must act as below.

- The current node is an initiator. If the sub-tree of this node is same as the network's size, the algorithm finishes. The current node broadcast a message with it's id as the leader. If the sub-tree is less than the network size, another round of the algorithm begins.
- The current node is passive. Then according to echo's algorithm, this node must return the received message to it's parent along with the length of it's sub-tree.

2.2 Code

This algorithm is implemented with the help of runner code provided with the project description.The runner code runs multiple nodes with the help of python multi-thread processing. Nodes are connected to each other with weighted edges which may or may not have packet loss or delay. Each node is implemented by node.py, and runs until end of the algorithm. Nodes communicate

with the help of *rabbitmq* [1] library in python. On every received message, perocess-msg function is called to handle the message according to the algorithm. Initially each node's id, round, parent, status, sub-tree length are set to -1, 0, -1, False, 0. Initial set-up is done at initiate function.

When receiving a message we act on two conditions, whether it's a new message with higher round/id or the message is a returning message from neighbors. In the first condition we change to this wave as referred in the figure1. By changing the wave, the current node's id becomes the new wave's id and it's parent will be the sender of the message also if the node is still active it becomes passive by changing it's status to *False*. The variable returned-messages-in-round keeps record of the messages with the same id and round received by the current node. Since this node has recently changed to the new wave this variable must be set to *empty*. On the other hand if a node has only one neighbor which will be it's parent, it must immediately return the received message to it's parent since no other node is left to send the message to.

```
44      if (message[0]> world.current_round) or (message[0] == world.current_round and message[1] > world.current_id) :
45          #  change to the received wave
46          world.current_parent = src
47          world.current_round = message[0]
48          world.current_id = message[1]
49          world.current_status = False
50          returned_msgs_in_round.clear()
51          world.sub_tree_length = 0
52          log(f'changed wave to {world.current_id}')
53          #  according to echo algortihm, when joining a wave a node must send
54          #       the received wave's message to it's neighbor except for the parent
55          for neighbor in neighbors :
56              if  neighbor != src :
57                  world.send_message( neighbor, [message[0], message[1], 0])
58          world.current_message_count += len(world.neighbors)
59          if (len(neighbors) == 1) :
60                  world.send_message( src, [message[0], message[1], 0])
61                  world.current_message_count += 1
```

Figure 1: Incoming message has a higher round/id.

If the incoming message has the same id and round as the current node then it must wait until receipt of the message from all the neighbors. When the current node has received message from all neighbors, if passive, the current node returns the message along with it's sub-tree length to it's parent according to lines 101-104 of figure2. If the current node is still active, it acts according to the length of it's sub-tree. The sub-tree of this node shows how many other nodes took part in it's wave. If the whole network took part in it's wave then all other nodes are passive and this node is the leader. If the sub-tree includes the whole network the current node broadcast a message with it's id and *exit* to the network and announces itself as the leader.

```
63      elif (message[0] == world.current_round) and (message[1] == world.current_id) :
64          if not src in returned_msgs_in_round and src != world.current_parent :
65              if src != world.current_node :
66                  #  adding returned sub_tree of child into the parent's sub_tree
67                  world.sub_tree_length += sub
68                  #  adding child to the sub_tree
69                  world.sub_tree_length += 1
70              returned_msgs_in_round.append(src)
71          log(f'received message from {returned_msgs_in_round}')
72          if (world.current_parent in temp_neighbors) :
73              temp_neighbors.remove(world.current_parent)
74          log(f'neighbors {temp_neighbors}')
75          #  according to echo's algorithm when the submitted wave's messages return from all neighbors a node must
76          #   a) send the message to it's parrent if it's not an initiator
77          #   b) decide if it's an initiator
78          if (set(returned_msgs_in_round) == set(temp_neighbors)) :
79              log("returned from all of the neighbors ")
80              if (world.current_status) :
81                  if (world.sub_tree_length + 1 >= world.network_number_of_nodes) :
82                      #  if the node's sub_tree length is the same as the network size,
83                      #       all nodes have received it's wave and joined it.
84                      #  since all nodes have joined the wave, all the network accept this node as the leader
85                      print(f' final leader is me {world.current_node} with the winning id as {world.current_id}', file= sys.stdout)
86                      for neighbor in neighbors :
87                          world.send_message(neighbor, "exit")
88                      world.current_message_count += len(world.neighbors)
89                      print(f'total message sent from this node is  {world.current_message_count}', file= sys.stdout)
90                      sys.exit()
91                  else :
92                      #  if sub_tree length is less than the network size another round of the algorithm must begin
93                      world.current_round +=  1
94                      world.sub_tree_length = 0
95                      world.current_id = random.sample(range(world.network_number_of_nodes), 1)
96                      returned_msgs_in_round.clear()
97                      for neighbor in neighbors :
98                          world.send_message( neighbor, [world.current_round, world.current_id, 0])
99                      world.current_message_count += len(world.neighbors)
100             else :
101                 #     a node that's not an initiator must return the message to it's parent when received from all neighbors
102                 log(f'returning to parent {world.current_parent}, the id is {world.current_id} and the sub tree length is {world.sub_tree_length}')
103                 world.send_message(world.current_parent, [world.current_round, world.current_id, world.sub_tree_length])
104                 world.current_message_count += 1
```

Figure 2: Incoming message has the same round and id.

3 Performance Analysis of the Algorithm

In this section we have generated random graphs using *networkx* library with different topologies to analyze the performance of the algorithm on each of them. As referred to in [2] message complexity of this algorithm is $\mathcal{O}(E)$. Since each message in a round is sent twice in each edge and we have finite number of rounds. Here we will test this claim with the results we got from the implementation.

3.1 Output on Rings

Each ring with N nodes has N edges. With the help of networkx generator function, several rings with different number of nodes was generated. The number of messages communicated in each graph, was plotted in figure3 as a function of the edges of the graph.

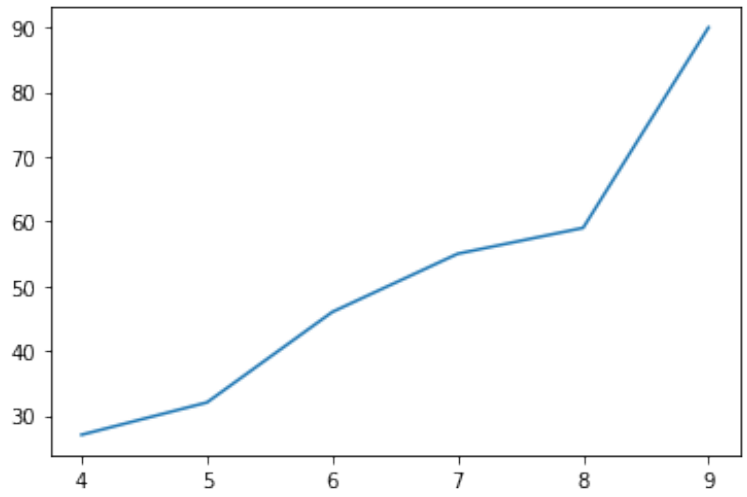

Figure 3: Number of communicated messages versus number of edges of the graph

3.2 Output on Random Trees

Each tree with N nodes has $N-1$ edges. Here several random trees were generated and testes, the result can be seen from figure4 as plotted. The x axis is the number of the edges of the tree and y axis shows the number of messages communicates until deciding the leader.

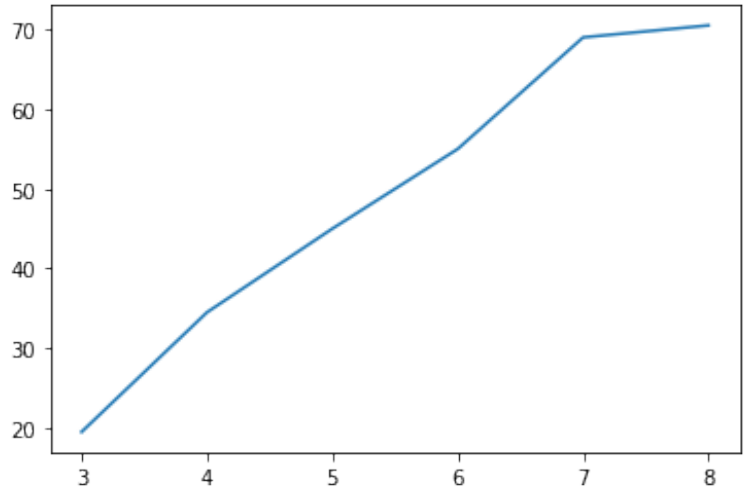

Figure 4: Number of communicated messages versus number of edges of the graph

3.3 Output on Complete Graphs

A complete graph of N nodes has $\frac{N \times (N-1)}{2}$ edges total. Figure 5 shows the number of messages as the number of edges on the graph increases.

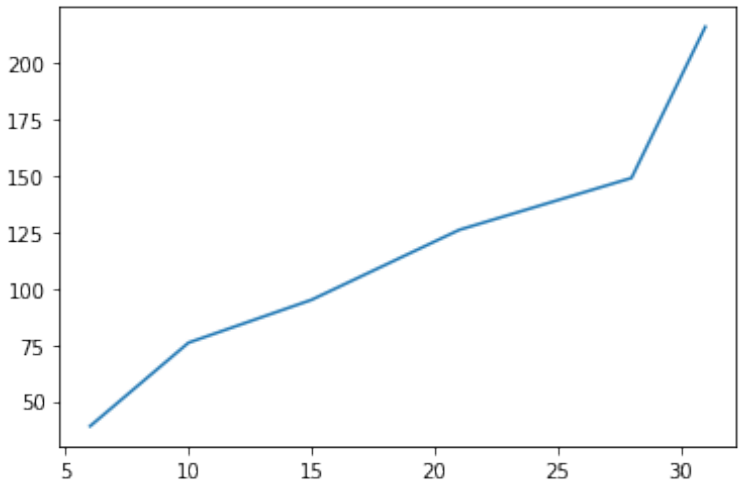

Figure 5: Number of communicated messages versus number of edges of the graph

3.4 Discussion

As we can see from the figures 3-5, results from our execution is also somewhat a linear function of the edges of the graph.

References

[1] https://www.rabbitmq.com/

[2] W. Fokkink, Distributed algorithms: an intuitive approach. 2018.