A Real Time Analysis of Leader Election Algorithms in Networks with Unknown Participants

Abstract: A network with unknown participants is simply a network where the processes have very primitive knowledge of the system. This type of network is synonymous to an address book, and introduces a factor of anonymity between processes, such that for any two nodes v, w if an edge exists (v,w), the converse edge (w,v) does not necessarily exist. In this report we investigate the leader election problem in general networks with unknown participants. Throughout this paper we will be analyzing and measuring the time and message complexity of the leader election algorithm identified in a research paper written by Dr. Jeremie Chalopin, Dr, Emmanuel Godard and Dr. Antoine Naudin [JEA14]. We will test whether it is possible to elect a leader in an anonymous network with the algorithm provided in the original paper, and if not why.

Introduction: Distributed systems are used throughout computer science and becoming increasingly dynamic. There have not been many studies on static models where the local connectivity does not evolve during the computation. Static models of networks are less realistic as networks today are becoming more dynamic and ad hoc, with the construction and destruction of these network occurring often.

Consider a set of participants that communicate with phones. Initially, each person knows a subset of the total phone registry. For example, Frank may have Ferns number, yet Fern may not know Franks number. In this situation Frank can contact Fern, but Fern cannot communicate with Frank until first receiving a message from him. Fern may not even know of Franks existence until first being contacted by him. During the preliminary stages of this network connectivity is low, but over time it

slowly increases the amount of communication between nodes by building channels, or adding a new contact into the address book.

In an arbitrary network, the underlying communication digraph is an directed graph denoted by $G =< V , E >$, where V is the set of vertices and E is the set of edges. Processes each have unique identifiers, and communicate by sending and receiving messages. Initially a process can send messages only to a subset of G, this defines the first directed digraph G^0. The contact list of a process v, $contact_v$, a list of processes that v has received a message from at least once, that will be extended whenever a message is received from an processes p where $id_p \notin contact_v$. When all nodes exchange their initial knowledge of the topology of the network, the possible communication digraph is isomorphic to the initial digraph G. The network is reliable, but asynchronous so the messages will always be delivered, without loss of generality we assume that message transaction is instantaneous and only a single message may be read per communication round.

Related Work. The network with unknown participants model presented here is a slight variation of a model that has been formally introduced and studied in [JEA14]. In [JEA14] there exists a necessary and sufficient condition that states if isolated executions are performed, it is possible for more than one processes to be elected. As stated in [JEA14] it is impossible, see a detailed proof within [JEA14], by the isolation lemma that a universal leader election algorithm be established. In the work done by the authors of [JEA14] the algorithm must use a family specific characteristic function to avoid disjoint isolated executions. This ensures nodes have adequate information about the network in order to decided to, or not to, become a leader.

The Model:

Message passing model: Processes communicate by sending and receiving messages over a restricted subset of the network. The edges of the network that link the vertices are directed and asynchronous. A message sent by an arbitrary node v is of the form $< id_v, SuperMailbox_v >$. The $SuperMailbox_v$ is a list containing tuples of the form $< id_u, Mailbox_u, status_u >$ where $status_u \in \{'leader', 'follower', 'undecided'\}$, $\forall u \in contact_v$.

Processes identities. Each process must be given a unique identifier in order to maintain its isolation from other processes. If two processes were to have the same identity the conditions for leader election would not be satisfied and the algorithm will not complete correctly, if at all.

Port labelling. Each vertex of the digraph will have a contact list of its known neighbours which is a subset of nodes in the network. When a node v receives a message from a neighbour node w, it receives the message through the port labelled w. If $id_w \notin contact_v$ then v adds w's id into its contact list.

Graph labelling. Initially the digraph contains $population$ many nodes, where the initial information that each node contains is at least their own *id*, and a list of successors, connected neighbours, which make up a digraph $H \subseteq\downarrow G$. For a digraph H which is closed on the successors of G denoted by $H \subseteq\downarrow G$, for a more detailed definition of the successor list see [JEA14]. The queue containing the messages in transit between nodes will begin as empty.

Distributed Algorithm. A distributed algorithm is a set of state transition rules. Such transition rules are function of the current local state of the system. In our setting, three kinds of transitions are possible for a process v : it can modify its state, it can receive a message from a neighbour or it can send a message to all its known neighbours [JEA14]. If a message is received from a process w such that $id_w \notin contact_v$ v updates $contacts_v$ with id_w . When a node sends a message, it sends the same message to all of its known neighbours using a send all function. This type of message passing floods the network with messages communicating to all processes in the current contact lists this grows the contact lists and more communication is initiated. Each node is essentially learning about other processes through neighbours it knows. Below is an example in an arbitrary network with a *population* of size five the second image will be after a few rounds of communication to see how the contact list has been built from the initial digraph.

Execution Representation. An execution of a leader election algorithm is a sequence of changes on vertices state within the graph. The nodes all start in a state of undecided leader status. After execution of communication rounds there is knowledge built in the network which allows each node to decide if it should be a leader or not. After sufficient knowledge is built should a node decided not to be a leader it will change its state to resemble this.

Algorithm Properties. An execution finishes when there is no progress that can be made in a nodes local algorithm and no message is in transit. In an execution, a process decides to alter its state from *undecided* only once during the execution. Execution ends if every process sets their corresponding states to either *leader* or *follower*.

Knowledge. As we will see, some problems need additional global information or knowledge to be solved. In this case we are operating on the family which knows the number of processes in the network, we will term this the *n-vertex* family. Therefore each processes in the network must know the number of vertices in the digraph and must also know the characteristic function of the *n-vertex* family

Remains Universal. A universal algorithm for leader election must work on every family of graphs. As shown in [JEA14] by the isolation lemma, universality of the leader election algorithms is impossible, and in the implementation provided by [JEA14] we avoid issues of universality by limiting the computation to specific families of graphs.

Experiments: We will analyze, in real time, the time complexity and message complexity of the leader election algorithm provided by [JEA14].

To analyze the leader election algorithm an average sample of completions must be taken in order to eliminate the variance between random generations. A sample of 10 random graphs has been chosen to represent the population for each test.

Each graph will contain a specific number of nodes per test, the first set of 10 tests will be run on simply 2 node digraph. When the 10 tests have been completed and times recorded, the average can then be compared for an arbitrarily large graph so we can find a correlation between population size and time.

Next the nodes of the graph will be increased to a population of 4, and the time test will then be run the same way. When all the times and averages are calculated for the graphs the averages can be again compared. Finally the timed tests will be run on graphs with population size of 10 and then compared with a theorized time expected to see how well our analysis was on the same graphs of smaller size. This is to see if we can predict a runtime of a network with any amount of nodes. Analyzing the data and providing the statistics will continue in the rest of the body of this report.

Discussion: For the network that we have been working with, each node has unknown neighbours. In order for us to simulate how the leader election algorithm would work with unknown neighbours we made it so that every node knows population size of the network.. This is a bit of an issue as this does not take into account extra time and message complexity, as well as the possibility of errors in establishing the ladder information. In our case we assume there will be no errors that occur to limit the variability in our results.

Each node will send at least one message to all nodes in the network, when that node decides to be a *leader* of *follower*. The receiving is the hard part, each time a node receives a message they change their state and have to inform everyone else of this change, which is why the number of

messages received is what affects the runtime and message count largely.

Results: It turns out that in a graph with only two nodes there is no variability in the amount of messages, only in the amount of communication rounds needed to complete the leader election. The reason we know there will only ever be eight messages sent in a two node graph is because there is no variability in how the graph is arranged. The distances between nodes, nodes will be in different positions, but in every iteration there will be one node that knows about the other and the other node will not know anything until after the first round of communication. So in every arrangement the same variables are present other than which node is qualified to be the leader.

Also on a network with two nodes each node can only send one message per tick, each communication round will consist of only two messages. When one node has decided not to be the leader, then one message and one response message will be sent to make sure that the other remaining node should be the one that will become the leader.

Proof:

The rounds of communication are what we will be using for the unit of time measurement. During an Average run of our simulation the message count is roughly the amount of nodes multiplied by the amount of communication rounds there are. So O(r*n) where r is the highest amount of ticks recorded, and n is the number of nodes in the graph. This is a very high approximate as curve can be better modeled by a $\sqrt{x}$ style function, we could not determine a rate of growth that matched closer to

the real time results then a linear function. The important thing now is how to find the highest time for graphs with nodes numbering n.

To find the upper bound on the communication rounds many test runs were applied to our simulation, times evaluated and recorded. Once the times were recorded for five different sized graphs the maximum communication rounds from each set of tests were used in looking for a pattern between them. Using a difference table a pattern was created and an algorithm formed that with a supplied size of a graph can find how many communication rounds it will take before the leader election algorithm is completed.

The algorithm formulated by this reverse engineering is $a_n = (1/6)(41n^3 - 132n^2 + 223n - 90)$

Where a_n is the highest possible communication rounds for a network of size n. With an arbitrary size network we can now compute the maximum expected communication rounds and messages sent during all of the communication rounds. In a network of size 9, the formula will be filled out as follows: $a_n = (1/6)(41(9)^3 - 132(9)^2 + 223(9) - 90)$. This is equal to 3519 ticks, or 3519 communication rounds in the network. See figure 1 below, it illustrates that the number of communication rounds is below the maximum. Also the maximum message count is computed as O(9*3519) messages. Therefore the upper bound on the running time of the algorithm is approximately $O(n^3)$.

Since each node is essentially flooding the graph with a message to every node in every communication round, each node will be sending at most one message per communication round. This

means that the message count is directly correlated with the expected runtime of the algorithm (less than maximum communication rounds)

Theorems: The runtime might change depending on when the leader is elected. The runtime is typically longer when the leader is the last node to be coloured, this is thought to happen because each node will have already decided that it will not be the leader, but each one must decide again after the leader has been elected.

Worst case scenario one node points to all other nodes in the graph, and no one points to it. All the other nodes decide that they are not the leader, and the one that was pointing to them all becomes the leader.

Methodology:

To come up with an algorithm used to find the time for a graph of any size we must find a pattern in the results from each size of network with known sizes. The communication rounds will be needed in order to find the maximum amount of messages possible. Pattern analysis between simulations will serve as a basis for our estimation algorithms.

Figures:

Figure 1 above.

Leader election, unknown neighbours 1 | Leader election, unknown neighbours 2

Leader election, unknown neighbours 3 | Leader election, unknown neighbours 4

Tables:

# of nodes	# of messages	Time
2	8	6 Ticks
2	8	4 Ticks
2	8	5 Ticks
2	8	4 Ticks
2	8	6 Ticks
2	8	5 Ticks
2	8	5 Ticks
2	8	5 Ticks
2	8	5 Ticks
Statistics	Mean = 8 Median = 8 Standard d = 0	Mean = 5 Median = 5 Standard d = 0.6667
# of nodes	# of messages	Time
3	49	22 Ticks
3	56	25 Ticks
3	52	23 Ticks
3	48	23 Ticks
3	40	20 Ticks
3	43	20 Ticks
3	39	19 Ticks
3	51	24 Ticks
3	49	22 TIcks

Statistics	Mean = 47.44 Median = 49 Standard d = 5.681	Mean = 22 Median = 22 Standard d = 2
# of nodes	# of messages	Time
5	616	162 Ticks
5	312	147 Ticks
5	610	218 Ticks
5	536	197 Ticks
5	578	213 Ticks
5	415	147 Ticks
5	474	170 Ticks
5	596	154 Ticks
5	559	143 Ticks
Statistics	Mean = 521.78 Median = 559 Standard d = 102.84	Mean = 173.33 Median = 162 Standard d = 29.453
# of nodes	# of messages	Time
4	202	67 Ticks
4	184	82 Ticks
4	202	67 Ticks
4	128	44 Ticks
4	128	44 Ticks
4	184	82 Ticks
4	211	70 Ticks
4	213	71 Ticks
4	130	61 Ticks
Statistics	Mean = 175.78 Median = 184 Standard d = 36.738	Mean = 65.33 Median = 67 Standard d = 13.89

Diagrams:

Conclusion:

This may not be the most efficient algorithm, but it works. For small networks of nodes this algorithm can complete in a relatively short amount of time. Growing the network to a size more than ten nodes comes with a large wait for completion. During our testing we had tested a fifteen node network, after six hours it had not even reached the highest amplitude of active messages which as seen in the diagrams above displaying a five node network the amplitude is only about at 25-30% of the total time the algorithm took to elect a leader.

References:

[JEA14] Jérémie Chalopin, Emmanuel Godard and Antoine Naudin. What Do We Need to Know to Elect in Networks with Unknown Participants?. In SIROCCO 2014, page 279-294.