Lecture Notes for CSCI 379: Distributed Computing

Beeping Deterministic Leader Election[1]

Förster, Seidel, Wattenhofer

Professor Talmage

February 7, 2022

---

1 Overview

The idea of the paper is to solve Leader Election in a very limited communication system. This is interesting to help further our understanding of the minimum level of communication required to solve interesting problems, and the system models low-power, low-resource robots.

Breaking down the title:

- Deterministic: This model is very hard to work in, so all prior algorithms for this problem are randomized.
- Leader Election: The problem under consideration.
- Multi-Hop: The communication graph is not complete: some nodes are at distance > 1 from each other.
- Beeping Networks: The communication model, which we'll discuss in more detail below.

So, what the paper does is show that it is possible to solve Leader Election deterministically in a general communication graph in the beeping model. Everything else is extra, such as:

- Optimality: They argue that their algorithm is nearly as efficient as the best randomized algorithms.
- Synchronized Wakeup: An extra feature that allows weakening the model slightly.
- Balanced Counters: While necessary for their solution, these are not a main result.

Except for balanced counters, the paper would be very nearly as strong without these. Discerning these differences can greatly reduce the cognitive load from reading a paper for the first time.

2 Model

While I'd typically counsel you to leave the model section for much later, once you've skimmed the content portion of the paper, in this case, it's worth getting at least a basic understanding of the system model, since it's quite different than what you're used to.

Exercise: Where does the paper fall in our grid of models? (synchrony, communication, failures)

- Synchronous: All processes are aware of the current round.
- Failure-free: Not explicitly mentioned, but that defaults to meaning that we're not handling errors.
- Uniform algorithm: No knowledge of n
- Non-anonymous: Processes have IDs, but they come from a range polynomial in n. This is one way to get around the problem of algorithms which have complexity dependent on IDs–log of a polynomial of n is just $O(n)$.
- Undirected graph: Unlike what we've seen before, processes do *not* know their own degree, and cannot distinguish neighbors.
- Beeping: Messages carry a single bit, and a node can only tell whether none of its neighbors sent a message or at least one did. Further, a node cannot receive *any* information in a round when it is sending.

Exercise: Can you solve LE with just two processes in this model? Can you do it in less than $O(i)$ rounds, where i is the smallest ID? What if you know that their IDs have different parity?

Exercise: Can you think of real-world system where such a model might apply?

3 High-Level Idea

Some of the challenges that need to be overcome:

- Forwarding messages is non-trivial, so we can't communicate across the entire graph.
- We can't just do time-division multiplexing when we don't even know how long IDs are, and they might be different lengths.
- If we want to build groups of processes which agree on a leader (similar to how we talked about building spanning trees), there are timing problems, since different groups may not agree on when to merge.
- With so little graph information, how do we know when the whole network agrees so we can return and top making so much noise?

Approach:

- Have processes compete in their 2-neighborhoods (with all nodes within distance 2)
- First run a protocol to determine the longest ID in competition, then just let the longest IDs compete in a much simpler algorithm since length is equal.
- As processes win local competitions, the processes they've beaten will compete on their behalf, increasing the range of the competition. Repeat this until the entire graph is working for a single node.

- Build a timing structure that doesn't allow any process to start a new, longer-range competition until its neighbors are ready to do so, as well.
- This timing structure will also allow a process to detect when its influence has spread across the entire graph. When enough time has elapsed without hearing that either it has been defeated or that it has defeated new nodes, a process concludes that it is leader, announces that, and terminates.

4 Round Breakdown Example

Most of the cleverness of the paper relates to showing that blocks of three rounds are enough to achieve useful communication. By bundling these together to get useful communication on different topics, then repeating a *lot* of times, we can accomplish our goals.

Consider first the algorithm for determining the length of the longest nearby (distance ≤ 2) ID. Fundamentally, we just need to keep listening until there are no more beeps, and then we know how long the longest ID among our neighbors is. The problem is that we cannot both beep and listen, so we need to combine these somehow, or everyone will sit silent and we'll learn nothing. (We can't have processes take turns beeping out the length of their ID, as that would require already solving leader election or something similar.) To accomplish both tasks, and coincidentally extend the range of long IDs' influence, the paper combines communication rounds in triples (one of the many things called *phases* in the paper): $[announce, relay, claim]$

- The *announce* round is for nodes who have not exhausted the length of their ID to communicate.
- The *relay* round is for nodes who have passed the length of their ID to relay beeps heard in the immediately preceding *announce* round to all their neighbors.
- The *claim* round is for nodes to claim that they have won. The timing of beeps here controls whether listening processes support a nearby node or are just disqualified.

So, each process beeps in the *announce* round as many times as there are bits in its ID, minus 1. At this point, a process will hear a beep in the *announce* round if and only if it has a neighbor process with longer ID. If it does, then it switches to supporting those processes (since it doesn't know how many might be longer, just that there's at least one), instead of itself.

4.1 Longest ID Logic

Let's dig deeper into the logic of the longest-ID competition. Each node will beep in round 0 for as many phases as there are bits in its ID (minus one), and relay those beeps in round 1 of each of those phases. Then, when a process gets to the end of its ID's length, interesting things happen. The idea is that a process wants to know if it has the longest ID in its 2-neighborhood, in which case it stays active, or if one of its neighbors does, in which case it becomes passive, supporting that neighbor, or in the third case, it becomes inactive, forming a barrier between active nodes.

- active: ID is longest in 2-neighborhood
- passive: Neighbor has ID longest in *this node's* 2-neighborhood
- inactive: No node in 1-neighborhood is longest in *this node's* 2-neighborhood, or separating active nodes with different lengths

The logic for the round right after a process finishes announcing is somewhat convoluted, so lets work through all the cases. Red cells indicate a process beeping while hearing nothing, which cannot happen, as a beeping process cannot hear whether its neighbors beeped or not (effectively, it will hear itself beep).

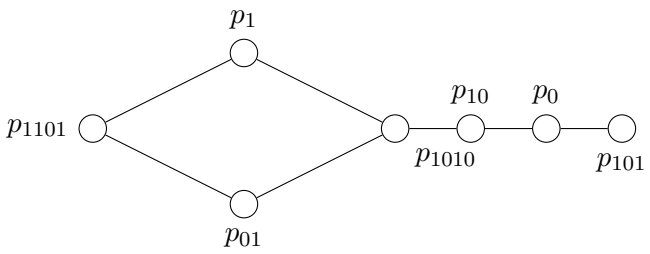
Hears	announces	relays	claims	resulting state
$[0,0,0]$	0	0	1	
$[0,0,1]$	0	0	1	active
$[0,1,0]$	0	0	0	inactive
$[0,1,1]$	0	0	0	inactive
$[1,0,0]$	0	1		
$[1,0,1]$	0	1		
$[1,1,0]$	0	1	0	passive
$[1,1,1]$	0	1	0	inactive

Figure 1: Possible outcomes in round l_{in} of the algorithm to determine the locally-longest IDs.

```
1: Phase 1 to l_in − 1
2:     Beep in round 0
3:     Beep in round 1
4: end Phase
5: Phase l_in
6:     if hear beep in round 0 then
7:         beep in round 1
8:         become passive
9:     else if hear beep in round 1 then
10:        become inactive
11:    else
12:        beep in round 2
13:        end as active
14:    end if
15: end Phase
16: Phase > l_in
17:    if hear beep in round 0 then
18:        beep in round 1
19:    else if hear beep in round 1 then
20:        become inactive
21:    else if hear beep in round 2 then
22:        end as passive
23:    else
24:        become inactive
25:    end if
26: end Phase
```

4.2 Example Run

Consider the following graph:



They will beep in the following pattern:

Round:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_{1101}	1	1	0	1	1	0	1	1	0	0	0	1(a)			
p_1	0	1(p)	0	0	1	0	0	1	0	0	0	0(p)			
p_{01}	1	1	0	0	1(p)	0	0	1	0	0	0	0(p)			
p_{1010}	1	1	0	1	1	0	1	1	0	0	0	1(a)			
p_{10}	1	1	0	0	1(p)	0	0	1	0	0	0	0(p)			
p_0	0	1(p)	0	0	1	0	0	0(i)							
p_{101}	1	1	0	1	1	0	0	0	1(a)						

NOTE: In class, we said we thought p_0 should be passive, not inactive, because it doesn't hear a beep from p_{101} in round 7. However, it hears a relay beep from p_{10} in round 8, which is a relay for which it heard no announcement, so it becomes inactive. It does this to act as a barrier between the active nodes with different length IDs.

4.2.1 Highest ID Competition

Now that processes have narrowed the field of active competitors to those which have longer IDs than any other process in their 2-neighborhoods, they can start to compare IDs to see which is larger. We can only compare IDs of the same length, as we need to compare bit-by-bit from highest-order to lowest, and comparing different length IDs would not align properly. Now, as soon as a node hears a beep (representing a 1) in a phase corresponding to a bit in its ID which is 0, it knows it is out.

Round:	l_{out}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_{1101} (a)	4	1	1	0	1	1	0	0	0	0	1	1	1			
p_1 (p)	4	0	1	0	0	1	0	0	0	0	0	1	0			
p_{01} (p)	4	0	1	0	0	0	0	0	0	0	0	1	0			
p_{1010} (a)	4	1	1	0	0	0(p)	0	0	0	0	0	0(i)				
p_{10} (p)	4	0	1	0	0	0	0	0	1	0	0	0	0(i)			
p_0 (i)	$\emptyset$															
p_{101} (a)	3	1	1	0	0	0	0	1	1	1						

Finally, now that we've separated nodes appropriately, active nodes beep their IDs in binary, and passive nodes store those IDs as the ID they are campaigning for.

4.3 Results of One Campaign

- Every active node has the largest ID in its 2-neighborhood.

- Neighbors of the highest ID in its 3-neighborhood stay active and adopt that ID. (This means that in subsequent round of campaigning, the largest IDs will continue to spread.)
- Each node takes $O(\ell)$ rounds, where ℓ is the length of the longest ID in its 1-neighborhood, which implies $O(\log n)$ for a complete round of campaigning.

5 Extending to Larger Results

The tricky part of running multiple rounds of campaigning is that different segments may want to move to the next round at different times. This is non-trivial to handle, since we can't include a round number in a message in this model. To get around this, the authors introduce a mod3 counter, which ensures that all neighboring nodes are at most one round apart. This counter uses phases of 6 communication rounds, so now we can run multiple rounds of campaigning, using the balanced counter to prevent a node from starting a new round until all its neighbors have finished the previous round.

- Each phase is now 9 rounds of communication: 6 for the counter, 3 for campaigning.
- Each process uses its out ID from one round of campaigning as its in ID for the next.
- After D (diameter) rounds of campaigning, every node will have the same out ID, agreeing that that process is leader.
- Problem: Nodes don't know D, so they cannot terminate.

While nodes cannot know the actual communication graph, they can simulate an *overlay network*, which is basically a reduced set of edges on the same network. Specifically, the paper shows how to do this by maintaining increasing chains of *depth* values, modulo 3, radiating out from every active node, eventually these will all combine to one tree, and when an active node stops receiving notifications of updates, it knows it has taken over the entire graph.

- As with the counter to allow multiple rounds, this system reduces efficiency. There are now 9 rounds of communication to every 1 round of leader election–complexity increased by a factor of 10!
- 10 and 3 are constants, though, so despite actually taking 30 times as many rounds as just a single campaign, the asymptotic complexity doesn't change: $O(D \log n)$.

References

[1] Klaus-Tycho Förster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks - (extended abstract). In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2014.