Principles of Distributed Computing
Roger Wattenhofer, Summer 2003

Chapter 2

Symmetry Breaking 2: Leader Election

Section 2.1: Anonymous Leader Election

(Sometimes good to have a special node: a “leader”)
(Remember 2-coloring algorithm in tree – who is the root?)
(Ring is drosophila of distributed computing. But also token ring standard.)

Problem 2.1 [Leader Election]: Each processor eventually decides whether it is a leader or not, subject to the constraint that exactly one is a leader.

Remark:

- More formally: processors are in one of three states: undecided, leader, not leader. Initially every process is in the undecided state. When leaving the undecided state, a processor goes into a terminated state (leader or not leader).

Definition 2.2 [Anonymous]: A system is anonymous if processors do not have unique identifiers.

Definition 2.3 [Uniform]: An algorithm is called uniform if the number of processors (“n”) is not known to the algorithm (to the processors, if you wish). If n is known, the algorithm is called non-uniform.

Whether or not a leader can be elected in an anonymous system depends whether the network is symmetric (ring, complete graph, complete bipartite graph, etc.) or asymmetric (star, single node with highest degree, etc.). We will now show that non-uniform anonymous leader election for synchronous rings is impossible. The idea is that in a ring symmetry can always be maintained.

Lemma 2.4: After round k of a deterministic algorithm A, each processor is in state s_k.

Proof by induction: All processors start in the same state. A round in a synchronous algorithm consists of the three steps sending, receiving, local computation. All processors send the same message(s), receive the same message(s), do the same local computation, and therefore end up in the same state.

Theorem 2.5 [Anonymous Leader Election]: Leader election in an anonymous ring is impossible.

Proof: With Lemma 2.4: If one processor ever decides to become a leader (or a non-leader), then every other does too, contradicting 2.1 (for more than one processor). This holds for non-uniform algorithms, and therefore also for uniform algorithms. This holds for synchronous algorithms, and therefore also for asynchronous algorithms.

Remarks:

- Sense of direction is the ability of processors to distinguish neighbor processors in an anonymous setting. In a ring, for example, a node can distinguish the clockwise and the counterclockwise neighbor. Sense of direction does not help in anonymous leader election.
- Theorem 2.5 also holds for other symmetric network topologies.
- The algorithm is not allowed to use randomization!

Section 2.2: Asynchronous Ring

In the following, we assume non-anonymity. Each node has a unique identifier (as in the previous chapter).

Algorithm 2.6 [Clockwise]: Every node v does the following:

- Node v sends a message with its identifier (for simplicity also "v") to its clockwise neighbor. (Remark: if node v already received a message "w" with w > v, then node v can skip this step; if node v receives its first message "w" with w < v, then node v will immediately send "v".)
- If v receives a message "w" with w > v, then v forwards "w" to its clockwise neighbor. Node v decides to not be the leader, if it has not done so already.
- If v receives a its own identifier "v", then v decides to be the leader.

Theorem 2.7 [Analysis of Algorithm 2.6]: Algorithm 2.6 is correct. The time complexity is O(n). The message complexity is O(n^2).

Proof: Let node z be the node with the maximum identifier. Node z will eventually send its identifier in clockwise direction, and since no other node can swallow the message "z", the message will arrive at z again – then z declares itself to be the leader. Every other processor will declare non-leader when forwarding message "z". Since there are n identifiers in the system, each processor will at most forward n messages, giving a message complexity of at most n^2. We start measuring the time when the first processor (the first that "wakes up") sends its identifier. For asynchronous time complexity (Chapter 1) we assume that each message takes at most one time unit to arrive at its destination. After at most n-1 time units the message therefore arrives at processor z, waking z up. Routing the message "z" around the ring takes at most n time units. Therefore processor z decides not after time 2n-1. Every other processor decides before processor z.

Remarks:

- Note that in Algorithm 2.6 nodes need to distinguish between clockwise and counter-clockwise neighbors. In fact they do not: It is OK to simply send your own identifier to any neighbor, and forward a message m to the neighbor you did not receive the message m from.
- If the identifiers of the nodes are ordered in the ring, then there are admissible (in fact, synchronous) executions where the message complexity is asymptotically tight.
- Can we achieve a better message complexity? Yes, with smarter forwarding.

Algorithm 2.8 [Radius growth, Ideas only]: (for detailed code see Attiya/Welch Alg. 3.1)

- Whenever a node v sees a message "w" with w > v, then the node decides to not be a leader and becomes passive.

- Active nodes search in an exponentially growing neighborhood (clockwise and counterclockwise) for nodes with higher identifiers, by sending out “probe” messages. A probe messages uses a power-of-two time-to-live (TTL); it is reflected when the TTL is zero (“reply” message). If there is no better candidate in the search area, then the node doubles the TTL and sends the next two probe messages (two neighbors). If there is a better candidate in the search area, then the node becomes passive.
- If a node v receives its own “probe v” message (not a reply), then v decides to be the leader.

Remark:
- This algorithm is asynchronous and uniform.

Theorem 2.9 [Analysis of Algorithm 2.8]: Algorithm 2.8 is correct. The time complexity is O(n). The message complexity is O(n log n).

Proof: Correctness is as in Theorem 2.7. The time complexity is O(n) since the node with maximum identifier z sends messages with round-trip times 2, 4, 8, 16, …, $2 \cdot 2^k$ with $k \leq \log n + 1$. Proving message complexity is slightly harder: if a node v manages to survive round r, then no other node in distance 2^r (or less) survives round r. That is, node v is the only node in its 2^r-neighborhood that tries round r+1. Since this is the same for every node, less than $n/2^r$ nodes try round r+1. Being active in round r costs $2 \cdot 2 \cdot 2^r$ messages. Therefore, round r costs at most $2 \cdot 2 \cdot 2^r \cdot n/2^{r-1} = 8n$ messages. Since there are only logarithmic many possible rounds, the message complexity follows immediately.

Remark:
- It is natural to ask whether one can design an algorithm with even less message complexity. We answer this question in the next section.

Section 2.3: Lower Bounds

(Lower bounds in distributed computing are often easier than in the standard RAM model because one can argue with messages that need to be exchanged.)

In this section we present a first lower bound. We show that Algorithm 2.8 is asymptotically optimal.

Definition 2.10 [Execution]: An execution of a distributed algorithm is a sorted list of events. An event is a record (time, node, type, message), where type is “send” or “receive”.

Remarks:
- We assume throughout this course that no two events happen at exactly the same time (or one can break ties arbitrarily).
- An execution of an asynchronous algorithm is generally not only determined by the algorithm but also by a “god-like” scheduler. If more than one message is in transit, the scheduler can choose which one arrives first.
- If two messages are transmitted over the same directed edge, then it is sometimes required that the message first transmitted will also be first received (“FIFO”).

For our lower bound, we assume the following model:

- We are given an asynchronous ring.
- We only accept uniform algorithms where the node with the maximum identifier can be the leader. Additionally, every node that is not the leader must know the identity of the leader. These two requirements can be dropped when using a more complicated proof.
- We play god and specify which message in transmission arrives. But we respect the FIFO conditions for links.

Definition 2.11 [Open Schedule] A schedule is an execution chosen by the scheduler. A schedule for a ring is open if there is an open edge in the ring. An open edge (undirected) is an edge where no message traversing the edge has been received so far.

The proof is by induction. First we show the base case:

Lemma 2.12: We are given a ring R with two nodes. We can construct an open schedule in which at least one message is received.

Proof: Let the two nodes be u and v with $u < v$. Node u must learn the identity of node v, thus receive at least one message. We stop the execution of the algorithm A as soon as the first message is received. (If the first message is received by v, bad luck for the algorithm!) Then the other edge in the ring (on which the received message was not transmitted) is open.

Since the algorithm needs to be uniform, maybe the open edge is not really an edge at all. Nobody can tell. We use this to glue two rings together, by breaking up this imaginary open edge and connect two rings by two (possibly imaginary) edges. We do this inductively.

Lemma 2.13: When gluing two rings with open schedules of size n/2 together, we can construct a ring of size n with an open schedule. For solving leader election, $\Omega(n)$ messages have to be exchanged.

Proof: Without loss of generality, sub-ring R_1 contains the maximum identifier. Thus each node in sub-ring R_2 must learn the identity of the maximum identifier, thus at least n/2 new messages must be received. The only problem is that we cannot connect the two sub-rings with both edges since the new ring needs to remain open. Thus only messages over one of the edges can be received. We play god (very much) and look into the future: we check what happens when we close only one of these connecting edges. With the argument that n/2 new messages must be received we know that there is at least one edge that will produce at least n/4 new messages when being scheduled. We schedule this edge, and leave the other open.

Theorem 2.14 [Asynchronous Leader Election Lower Bound]: Any uniform leader election algorithm for asynchronous rings has $\Omega(n \log n)$ message complexity.

Proof: $M(n) \geq 2\, M(n/2) + n/4$ (Lemma 2.13), with $M(2) \geq 1$ (Lemma 2.12).

Section 2.4: Synchronous Ring

The lower bound relied on delaying messages for a very long time. Since this is impossible in the synchronous model, we might get a better message complexity there.

Idea: In the synchronous model, not receiving a message is information!

First we have some additional assumptions:

- We assume that the algorithm is non-uniform (the ring size n is known)
- We assume that every node starts at the same time
- The node with the minimum identifier becomes the leader; identifiers are integers.

Algorithm 2.15 [Synchronous Leader Election]: Each phase consists of n time steps. The nodes count the phases, starting with 0. If, at the beginning of phase v node v has not received as message, v decides to be the leader and sends a message around the ring.

Remarks:

- Message complexity is indeed n.
- But time complexity is huge! If m is the minimum identifier it is $m \cdot n$.
- The synchronous start and the non-uniformity assumptions can be dropped, by using the wake-up technique, and by letting messages travel slowly.
- Bit complexity: Try to have only short messages, and count the total number of bits used.
- Several synchronous lower bounds: comparison-based algorithms (or if the time complexity cannot be a function of the identifiers) have message complexity Ω(n log n).
- We (implicitly) learn leader election algorithms for other graphs in the next chapter.