Section 8
Leader Election in Rings

The Leader Election Problem

• Each process should eventually decide that it is either the leader or it is not the leader.
• Exactly one process should decide that it is the leader.
• The leader process may be responsible for achieving synchronization in future activities of the system:
  • token re-creation
  • recovery from deadlock
  • play the role of the root node in the construction of a spanning tree, etc.
The Leader Election Problem – More formally

• An algorithm is said to solve the leader election problem if it satisfies the following conditions:
  - The terminated states are partitioned into elected and not-elected states. Once a process enters an elected (respectively, not-elected) state, its transition function will only move it to another (or the same) elected (respectively, not-elected) state.
  - In every admissible execution, exactly one process (the leader) enters an elected state and all the remaining processes enter a not-elected state.

The Leader Election Problem

Assumptions

• Ring topology
• The n processes have a notion of left and right.
  - For every i, 1 ≤ i ≤ n, p_i’s channel to p_{i+1} is labeled 1, also known as left or clock-wise, and p_i’s channel to p_{i-1} is labeled 2, also known as right or counter-clock-wise (addition and subtraction here are modulo n).
Model – Rings

- An algorithm is **anonymous** if the processes do not have unique identifiers that can be used by the algorithm.
  - Every process has the same state machine.
- Otherwise, the algorithm is called **eponymous** (or **non-anonymous**).
- If \( n \) is not known to the algorithm, the algorithm is called uniform
  - The algorithm looks the same for every value of \( n \).
- In an anonymous non-uniform algorithm, for each value of \( n \), there is a single state machine, but there can be different state machines for different ring sizes.
  - \( n \) can be explicitly present in the code.

Leader Election in Anonymous Synchronous Rings

**Theorem:** There is no non-uniform anonymous algorithm for leader election in synchronous rings.

**Lemma:** For every round \( k \) of the admissible execution of an anonymous leader election algorithm in a ring, the states of all the processors at the end of round \( k \) are the same.

**Proof:** By induction on \( k \).

- **Base case:** Straightforward since all processes begin in the same state.
- **Induction Hypothesis:** Assume the lemma holds for round \( k-1 \).
- **Induction Step:** Since all processes are in the same state in round \( k-1 \), they all send the same messages \( m_l \) to the left and \( m_r \) to the right.
- In round \( k \), all processes receive message \( m_r \) on its left edge and \( m_l \) on its right; because they execute the same program, they are in the same state at the end of round \( k \).
Leader Election in Eponymous Asynchronous Rings

An $O(n^2)$ Algorithm

Description of the algorithm:

• Each process sends a message with its identifier to its left neighbor and then waits for messages from its right neighbor.
• When is receives such a message, it checks the identifier in the message:
  - If it is greater than its own identifier, it forwards the message to the left.
  - Otherwise, it shallows the message.
• If a processor receives a message with its own identifier, it declares itself a leader by sending a termination message to its left neighbor and terminating.
• A processor that receives the termination message, forwards it to the left and terminates as non-leader.

Communication Complexity?

• No process sends more than $n$ messages.
• Is there an execution at which $\Theta(n^2)$ messages are sent?
An Algorithm with Communication Complexity $O(n \log n)$ - Main Ideas

- The $k$-neighborhood of a process $p_i$ in the ring is the set of processes that are at distance at most $k$ from $p_i$ in the ring (either to the left or to the right).

Main Ideas

- The algorithm works in phases:
  - $k^{\text{th}}$ phase, $k \geq 0$: a process tries to become a winner for the phase; a process becomes a winner if it has the largest id in its $2^k$-neighborhood.
  - Only processes that are winners in the $k^{\text{th}}$ phase continue to compete in the $(k+1)^{\text{st}}$ phase.

An Algorithm with Communication Complexity $O(n \log n)$ - Description

- In phase $k$, a process $p_i$ that is a phase $k-1$ winner sends <probe> messages with its identifier to the $2^k$-neighborhood (one in each direction).
- A <probe> is shallowed by a processor if it contains an identifier that is smaller than its own identifier.
- If the message arrives at the last process in the neighborhood, then that last process sends back a <reply> message to $p_i$.
- If $p_i$ receives replies from both directions, it becomes a phase $k$ winner, and it continues to phase $k+1$.
- A processor that receives its own <probe> message terminates the algorithm as the leader and sends a termination message around the ring.
An Algorithm with Communication Complexity $O(n \log n)$

**Pseudocode**

Algorithm 5 Asynchronous leader election: code for processor $p_i$, $0 \leq i < n$.

Initially, $asleep = true$

1: upon receiving no message:
2: if $asleep$ then
3:   $asleep := false$
4: send $(\text{probe}, id, 0, 1)$ to left and right

5: upon receiving $(\text{probe}, j, k, d)$ from left (resp., right):
6: if $j = id$ then terminate as the leader
7: if $j > id$ and $d < 2^k$ then // forward the message
8: send $(\text{probe}, j, k, d + 1)$ to right (resp., left) // increment hop counter
9: if $j > id$ and $d \geq 2^k$ then // reply to the message
10: send $(\text{reply}, j, k)$ to left (resp., right) // if $j < id$, message is swallowed

11: upon receiving $(\text{reply}, j, k)$ from left (resp., right):
12: if $j \neq id$ then send $(\text{reply}, j, k)$ to right (resp., left) // forward the reply
13: else // reply is for own probe
14: if already received $(\text{reply}, j, k)$ from right (resp., left) then
15: send $(\text{probe}, id, k + 1, 1) \Leftarrow le_{\sigma(k) \wedge \Delta(k)}$ // phase $k$ winner

- A message of type $<\text{probe}>$ contains the id $j$ of the process that sends it, the phase number $k$ and a hop counter $d$.
- A message of type $<\text{reply}>$ contains the id $j$ and the number of the current phase $k$. 

P. Fatourou, CS486 – Principles of Distributed Computing
An Algorithm with Communication Complexity $O(n \log n)$ - Analysis

- **Lemma:** For each $k \geq 0$, the number of processes that are phase $k$ winners is at most $n/(2^k+1)$.
- **Proof:**
  - Between two winners of phase $k$ there are $2^k$ other processes in the ring.
- **Remarks**
  - There is just one winner after $\log(n-1)$ phases.
  - The total number of messages is:
    
    \[
    5n + \sum_{k=1}^{\lceil \log(n-1) \rceil + 1} 4 \times 2^k \times n/(2^k+1) < 5n + 8n(\log n + 2)
    \]
- **Theorem:** There is an asynchronous leader election algorithm whose message complexity is $O(n \log n)$.

Leader Election in Synchronous Rings

- The reception of no message in a round is a piece of information. Does this help?
- **An $O(n)$ Upper Bound**
- **The Non-Uniform Algorithm**
  - Elects the processor with the minimal identifier as the leader.
  - It works in phases, each consisting of $n$ rounds.
  - In phase $i \geq 0$, if there is a processor with id $i$, it is elected as a leader and the algorithm terminates.
  - Phase $i$ includes rounds $ni+1$, $ni+2$, ..., $ni+n$.
  - At the beginning of phase $i$, if a process has id $i$, and it has not terminated yet, the process sends a message around the ring and terminates as a leader.
  - If the process does not have id $i$, and it receives a message in phase $i$, it forwards the message and terminates as the non-leader.
Bibliography

These slides are based on material that appears in the following book:

• H. Attiya & J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, Morgan Kaufmann, 1998 (Chapter 3)