Leader Election in Rings

- Under different models:
  - Synchronous vs. asynchronous
  - Anonymous vs. non-anonymous (knowledge of unique id)
  - Knowledge of “n” (non-uniform) vs. no knowledge (uniform)

- Impossibility result: there is no synchronous, non-uniform algorithm if the processors are anonymous.
  - Implies that there are no uniform algorithms as well
  - Implies that there are no asynchronous algorithms as well
Outline

- Leader election in asynchronous rings:
  - An \( O(n^2) \) messages algorithm
  - An \( O(n \log n) \) messages algorithm
- Brief mention of a lower bound
- Synchronous model:
  - Breaking the \( O(n \log n) \) barrier by abusing the synchronous model
  - For both uniform and non-uniform systems
- Leader election in arbitrary topologies
  - Using simultaneous DFS traversals

Asynchronous model: simple algorithm

Upon receiving no message:
\[
\text{send } \text{my\_id} \text{ in clockwise direction}
\]

Upon receiving "m"
\[
\text{case}
\begin{align*}
\text{m\_id} &< \text{my\_id}: \text{send } m \text{ in clockwise direction} \\
\text{m\_id} &> \text{my\_id}: \text{discard } m \\
\text{m\_id} &= \text{my\_id}: \\
&\quad \text{leader} = \text{my\_id} \\
&\quad \text{send } \langle \text{terminate, my\_id} \rangle \text{ in clockwise direction} \\
&\quad \text{terminate}
\end{align*}
\]

Upon receiving \( \langle \text{terminate, id} \rangle \):
\[
\text{leader} = \text{id}; \\
\text{send } \langle \text{terminate, id} \rangle \text{ in clockwise direction} \\
\text{terminate}
\]
Complexity

- Time complexity: $O(n)$
- Message complexity:
  - Clearly less than $n^2$ messages are sent
  - And $\Omega(n^2)$ is sent in the following worst case scenario

Hirschenberg-Sinclair Algorithm

- For bidirectional, asynchronous rings: achieve a $O(n \log n)$ message complexity
- Each node operates in phases:
  - In each phase, nodes that are still active send out their uid in both directions
  - In phase $k$, the tokens travel a distance of $2^k$ and return back to their points of origin
  - A token might not make it back if it encounters a node with lower uid
  - A node makes it to the next phase only if it receives its tokens back from the previous round
Detailed Description

Upon receiving no message
if asleep then
  asleep = false
  phase = 0
  send [my-id, out, 1] to left and right

Upon receiving [id, out, h] from left:
Case
  id < my-id and h > 1: send [id, out, h-1] to right
  id < my-id and h==1: send [id, in, -] to left
  id = my-id: leader = my-id

Upon receiving [my-id, in, -] from left and right:
  phase = phase + 1
  send [my-id, out, 2^phase] to left and right

Upon receiving [not-my-id, in, -] from left or right:
  send [not-my-id, in, -] to right or left

Correctness

- Messages from node with lower id is never discarded
- Messages from nodes with higher id eventually reach the node with the lowest id and gets discarded
- Therefore the correct leader is elected (safety)
- Liveness: eventually the node with the lowest id reaches phase log(n) and sends its id throughout the entire ring
**Communication Complexity**

- In phase 0, every processor sends a message:
  - Maximum of 4n messages

- In phase k+1:
  - Only processors that send tokens are those that “won” in the previous phase
  - There is at most one winner for every $2^k + 1$ processors
  - Winners after phase $k$: $\frac{n}{2^k + 1}$
  - Tokens travel a distance in phase $k+1$ of: $2^{(k+1)}$
  - Total number of messages in phase $k+1$:
    $$4 \cdot 2^{(k+1)} \cdot \frac{n}{2^k + 1} < 16n$$
  - Total number of phases: $1 + \log n$
  - Number of messages: $O(n \log n)$

**Question:**

- What is the time complexity?
Lower bound

AW has a lower bound proof:
- Asynchronous networks require $O(n \log n)$ messages to perform leader election

Proof sketch:
- Provide a lower bound for a constrained leader election problem:
  - Elects the node with the minimum id
  - Everyone should know the identity of the winner
- Construct an “open schedule” for a ring:
  - Open schedule is not complete; it is a prefix of an admissible execution
  - Open execution corresponds to taking a ring, blocking one of its channels, but allowing all other events to proceed as normal

Lower bound (contd.)

AW prove the following:
- Every ring of size $n$, has an open schedule that sends at least the following number of messages $M(n)$
- When $n=2$, $M(n) = 1$ (easy to show)
- For higher $n$, $M(n) = 2 M(n/2) + \frac{1}{2} (n/2 - 1)$
  - Assume that an open schedule exists for $n/2$ sized rings
  - Then show that there is an open schedule for $n$-sized rings:
  - The two scenarios are not distinguishable

- Wait for the two rings to reach a quiescent state
- Show that a further $\frac{1}{2} (n/2 - 1)$ messages will be sent if one of the two channels is unblocked
Announcements

- Will post some “homework” questions on chapter 2 from AW
- Send me email if you are still looking for a partner

Synchronous rings

- Leader election with fewer than $O(n \log n)$ messages is possible
  - Can convey information by not sending a message:
    “if you do not hear from me, then assume that …”
- Assume that:
  - Uids are positive integers
  - Can be manipulated using arbitrary arithmetic operations
- Two algorithms: TimeSlice, VariableSpeeds
- TimeSlice:
  - $n$ is known to all processors (non-uniform)
  - Unidirectional communication is sufficient
  - $O(n)$ messages
TimeSlice Algorithm

- Recall that a round in synchronous networks is:
  - Deliver all messages, have every processor take one compute step
- Define the notion of a phase
  - Each phase consists of “n” rounds
  - In phase k >= 0
    - If no one is elected yet
    - Processor with uid k:
      - Declares itself as the leader
      - Sends token with its uid around

- Message complexity: n
- Time complexity: n*(minimum uid value)

VariableSpeeds Algorithm

- Uniform algorithm: “n” is not known
- Unidirectional communication is sufficient
- Still achieves the O(n) messages bound
- Assumes there are two kinds of processors:
  - Those that are awake and participating in the leader election
  - Those that are non-participants and simply serve as relays
- Message life cycle:
  - A message is in phase one:
    - Until it is received by an awake processor
    - Forwarded immediately
  - A message is in phase two:
    - Once received by an awake processor
    - Forwarded after (2^message-uid – 1) rounds
Algorithm (contd.)

- When participant receives a message
  - If `message.id > my-uid` or the minimum message-id seen so far:
    - Swallow it
  - Else:
    - Delay for $2^{(message.id - 1)}$ rounds
- For a relay:
  - If `message.id > minimum message-id seen so far`, swallow it
  - Else, delay for $2^{(message.id - 1)}$ rounds
- If a processor gets its message back, it elects itself as the leader
- Correctness:
  - No processor will swallow the message with minimum uid
  - A message has to go through all processors before a leader is elected

Complexity

- By the time $UID_{\text{min}}$ goes around the ring, the second smallest UID has gone only half way, third smallest a fourth of the way, etc.
- Forwarding the token carrying $UID_{\text{min}}$ has caused more messages than all the other tokens combined

- Message complexity: $O(n)$
- Time complexity: $n \times 2^{UID_{\text{min}}}$
General Networks

What if a network has arbitrary topology?

Here is a simple algorithm based on DFS algorithm

- DFS algorithm from a specified root:
  - When a node first receives a message M
    - Send accept to sender
    - For each child:
      - Send M
      - Wait for accept or reject before considering next child
  - When a node later receives a message M
    - Send reject to sender

General networks (contd.)

- Start DFS spanning tree algorithm from all nodes
- In addition:
  - Send node’s uid along with M
  - When two DFS traversals collide, the copy with the lower uid wins
  - The other DFS stalls – no response is sent to the sender
    - The sender waits forever
  - Only the processor that has the minimum uid gets a response from all of its children
- Message complexity: $O(n \times m)$
- Time complexity: $O(m)$
- See text for details