Outline

- Specification of Leader Election
- YAIR
- Leader election in asynchronous rings:
  - An $O(n^2)$ algorithm
  - An $O(n \log(n))$ algorithm
- The revenge of the lower bound!
- Leader election in synchronous rings
  - Breaking the $\Omega(n \log(n))$ barrier

The LCR Algorithm

LeLann (1977), Chang and Roberts (1979)

- unidirectional
- asynchronous
- non-anonymous: every process has uid
- uniform (does not depend on n)

Upon receiving m from right
3: case
4: $m.uid > uid_i$
5: send m to left
6: $m.uid < uid_i$
7: discard m
8: $m.uid = uid_i$
9: leader := i
10: send <terminate, i> to left
11: terminate
12: endcase

Upon receiving no message
13: upon receiving <terminate, i> from right
14: leader := i
15: send <terminate, i> to left
16: terminate

Correctness

- messages from process with highest ID are never discarded
- therefore the correct leader is elected
- no other processor ID can traverse the entire ring
- therefore no one else is elected

Complexity

Message complexity: $O(n^2)$

This bound is tight...

Time complexity: $O(n)$

Can we do better?
The HS algorithm

- Ring is bidirectional
- Each process \( p_i \) operates in \( \text{phases} \)
- In each phase \( r \), \( p_i \) sends out "tokens" containing \( uid_i \) in both directions
- Tokens are intended to travel distance \( 2^r \) and return to \( p_i \)
- However, tokens may not make it back
  - Token continues outbound only if greater than tokens on path
  - Otherwise discarded
  - All processes always forward tokens moving inbound

All processes always forward tokens moving inbound

Correctness

Same as LCR:
- messages from process with highest ID are never discarded
- therefore the correct leader is elected
- no other processor ID can traverse the entire ring
- therefore no one else is elected

Communication Complexity

- Every processor sends a token in phase 0
- \( 4n \) messages
- For phase \( r > 0 \):
  - the only processors to send a token are those who "won" in phase \( r-1 \)
  - There is a winner for every \( 2^{r-1} \) processors (at most)
  - Winners in phase \( r > 0 \) - Tokens travel distance \( 2^r \)
  - Total number of messages sent in phase \( r > 0 \) is bounded by \( 4^r \) messages
- Total number of phases
- No. of messages bound by \( n(1+\log n) \) which is \( O(n \log n) \)

The Protocol

0: Init: asleep := true
1: upon receiving no message
2: if asleep then
3: asleep := false
4: send <probe, uid_i, 1, 0> to L and R
5: upon receiving <probe, uid_j, r, d> from L (resp. R)
6: if \( uid_j = uid_i \) then
7: leader := p_i
8: terminate
9: if \( uid_j > uid_i \) and \( d < 2^r \) then
10: send <probe, uid_j, r, d+1> to R (resp. L)
11: if \( uid_j > uid_i \) and \( d \geq 2^r \) then
12: send <reply, uid_j, r> to L (resp. R)
13: upon receiving <reply, uid_j, r>
14: if \( uid_j \neq uid_i \) then
15: send <reply, uid_j, r> to R (resp. L)
16: else
17: if already received <reply, uid_j, r> from R (resp. L)
18: send <probe, uid_i, r+1, 0> to L and R
Time Complexity

- Time for each phase: $2 \cdot 2^r = 2^{r+1}$
- Final phase takes $\rho$ (tokens only traveling outbound)
- Next to last phase is $r = \lceil \log n \rceil$
- Total time complexity excluding last phase $2 \cdot 2^{\lceil \log n \rceil}$
- Time complexity is at most $5n$ to $5n$

The revenge of the lower bound

So far we have seen:
- a simple $O(n^2)$ algorithm
- a more clever $O(n \log n)$ algorithm
- focus on message complexity

Facts:
- $\Omega(n \log n)$ lower bound in asynchronous networks
- $\Omega(n \log n)$ lower bound in synchronous networks when using only comparisons

Outline

- Specification of Leader Election
- YAIR
- Leader election in asynchronous rings:
  - An $O(n^2)$ algorithm
  - An $O(\log(n))$ algorithm
- The revenge of the lower bound!
- Leader election in synchronous rings
  - Breaking the $\lceil \log(n) \rceil$ barrier
  - The rise and fall of randomization

Leader Election with fewer than $O(n \log n)$ messages

- Synchronous rings
- UID are positive integers
- Can be manipulated using arbitrary arithmetic operations

<table>
<thead>
<tr>
<th>TimeSlice</th>
<th>VariableSpeeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$ is known to all processors</td>
<td>$n$ is not known to all processors</td>
</tr>
<tr>
<td>unidirectional communication</td>
<td>unidirectional communication</td>
</tr>
<tr>
<td>$O(n)$ messages</td>
<td>$O(n)$ messages</td>
</tr>
</tbody>
</table>

What about Time complexity?
What is special about synchronous rings?

- Can convey information by *not* sending a message
  
  “when your phone doesn’t ring, it’s me”

---

**TimeSlice**

Runs in phases
- each phase consists of $n$ rounds
  - if no one elected yet
  - processor with id $i$
  - declares itself the leader
  - sends token with its UID around

Message complexity: \( n \)

Time complexity: \( n \cdot \text{UID}_{\text{min}} \)

---

**VariableSpeeds**

- Each process $p_i$ initiates a token
- Different tokens travel at different speeds:
  - for token carrying UID, 1 message every \( 2^{\text{UID}} \) rounds
  - (each process waits \( \frac{2^{\text{UID}}}{2^i} \) rounds after receiving the token before sending it out)
- Each process keeps track of smallest UID seen
- Discard token with UID greater than smallest UID

---

**Complexity Analysis**

- By the time UID goes around the ring, the second smallest UID has gone only at most half way, third smallest at most a fourth of the way, etc.
- Forwarding the token carrying UID has caused more messages than all the other tokens combined

Message complexity bound by \( 2^m \)

Time Complexity \( n \cdot 2^{\text{UID}_{\text{max}}} \)