Network Algorithms

Leader Election
Leader Election

Nodes in network agree on exactly one leader. All other nodes are followers.

Reasons for electing a leader?

Reasons for not electing a leader?
Motivation

Reasons for electing a leader?

– Once elected, *coordination* tasks may become simpler
– For example: wireless medium access (break symmetry)

Reasons for *not* electing a leader?

– Reduced parallelism?
– Self-stabilization needed: re-election when leader „dies“
– Leader *bottleneck* / single point of failure?
How to elect a leader in a ring?
Model „Synchronous Local Algorithm“: Round

Send...

... receive...

... compute.
Anonymous Ring

Anonymous System
Anonymous nodes do not have identifiers.

Theorem
In an anonymous ring, leader election is impossible!

Why?
Theorem

In an anonymous ring, leader election is impossible!

First, note the following lemma:

Lemma

After round $k$ of any deterministic algorithm on an anonymous ring, each node is in the same state $s_k$.

Proof idea?!

By induction: all nodes start in same state, and each round consists of sending, receiving and performing local computations. All nodes send the same messages, receive the same messages, and do the same computations. So they always stay in same state...

QED

So when a node decides to become a leader, then all others do too.
Discussion

What is the basic problem?

Symmetry.... How could it be broken?

- How to elect a leader in a star?
- Randomization?
- What if nodes have IDs?
Asynchronous Ring

Let’s assume:
- non-anonymous nodes with unique IDs
- asynchronous ring (asyn start and transmissions)
- uniform ring: n unknown!
- no message losses etc.

How to elect a leader now?
Asynchronous Ring

Let’s assume:

- non-anonymous nodes with unique IDs
- asynchronous ring

Algorithm Clockwise

each node v does the following:

- v sends a message with its ID v to clockwise neighbor (unless v already received a message with ID w>v)
- if v receives message w with w>v then
  • v forwards w to clockwise neighbor
  • v decides not to be the leader
- else if v receives its own ID v then
  • v decides to be the leader

How to evaluate?
Criteria?
Asynchronous time?!
**Evaluation**

**Time Complexity**
Number of rounds. For asynchronous, assume max delay of one unit (of course no bound known to nodes).

**Message Complexity**
Number of messages sent.

„**Local Complexity“**
Local computations...

For our algorithm?!
Theorem

Algo is correct, time complexity O(n),
message complexity O(n^2).

Proof idea?

**Correctness:** Let z be max ID. No other node can swallow z‘s ID, so z will get the message back. So z becomes leader. Every other node declares non-leader when forwarding z (the latest!).

**Message complexity:** Each node forwards at most n messages (n IDs in total).

**Time complexity:** Message circles around cycle (depending on model, at most twice: once to wake up z, and then until z becomes leader).

QED

Can we do better?! Time? Messages? ...
Each node $v$ does the following:

- Initially, all nodes are active (can still become leader).
- Whenever a node $v$ sees a message with $w>v$, it decides not to be a leader and becomes passive.
- Active nodes search in an exponentially growing neighborhood (clockwise and counterclockwise) for nodes with higher IDs by sending out probe messages:
  - A probe includes sender’s ID, a leader bit saying whether original sender can still become a leader, and TTL (initially $=1$).
  - All nodes $w$ receiving a probe decrement TTL and forward to next neighbor; if $w$’s ID is larger than original sender’s ID, the leader bit is set to zero. If TTL=0, return message to sender (reply msg) including leader bit.
  - If leader bit is still 1, double the TTL, and two new probes are sent (for both neighbors); otherwise node becomes passive.
  - If $v$ receives its own probe message (not the reply): it becomes leader.
Am I leader here?
Am I leader here?
Am I leader here?

How to analyze?
Complexities?
Theorem

Algo is correct, time complexity $O(n)$, message complexity $O(n \log n)$.

Proof idea?

**Correctness:** Like clockwise algo.

**Time complexity:** $O(n)$ since node with max identifier sends messages with round trip times $2, 4, 8, \ldots, 2^k$ with $k \in O(\log n)$. The sum constitutes a geometric series and is hence linear in $n$.

**Message complexity:** Only one node can survive phase $p$ that covers a distance of $2^p$. So less than $n/2^p$ nodes are active in phase $p+1$. Being active in round $p$ costs roughly $2^p$ messages, so it’s around $O(n)$ per round over all active nodes. As we have a logarithmic number of phases, the claim follows.

QED
Can we do better?! 

Or how can we prove that we cannot? 

Lower bounds!
In message passing systems, lower bounds can often be proved by arguing about messages that need to be exchanged!

Concepts:
1. Generally, we need some definitions to characterize the class of algorithms for which the lower bound holds.
2. Moreover, in distributed systems, a (hypothetical) scheduler determines sequence of events...

An execution of a distributed algorithm is a list of events, sorted by time. An event is a record (time, node, type, message) where type is „send“ or „receive“.
Assumptions:

- Asynchronous ring: nodes \textit{wake up} at arbitrary times but always when receiving a packet
- nodes have IDs, and node with \textit{max ID} should become leader (strong assumption?)
- every node must know ID of leader
- uniform algorithm: \( n \) is not known
- arbitrary scheduler but links are FIFO

For our lower bound proof, we define the concept of \textit{open schedules}:

**Open Schedule**

Schedule chosen by scheduler. \textit{Open} if there is an \textit{open edge} in the ring. Edge is \textit{open} if no message traversing this edge has been received so far.

Note: any leader election algorithm must send over each edge at some point! Otherwise whole network could be hidden behind it.
Some Intuition...

Open Schedule

Schedule chosen by scheduler. Open if there is an open edge in the ring. Edge is open if no message traversing edge has been received so far.

Intuition: Open schedule = endpoints have not heard anything from nodes on this edge, protocol cannot stop yet as it may hide critical infos on the leader!

We want to show that there exists a bad schedule which requires lots of messages until a leader is elected. To achieve this, we play god and compute a bad open schedule inductively (looking into the future, where many messages will be sent!).
Proof by induction:

**Lemma: 2-node Ring**

Given a ring $R$ with two nodes, we can construct an open schedule in which at least one message is received. The nodes cannot distinguish this schedule from one on a larger ring with all other nodes being located where the open edge is.

Proof of Lemma: $u$ and $v$ cannot distinguish between the two scenarios!

How to make an open schedule?
Proof of Lemma: Open Schedule

Lemma: 2-node Ring

Given a ring \( R \) with two nodes, we can construct an open schedule in which at least one message is received. The nodes cannot distinguish this schedule from one on a larger ring with all other nodes being where the open edge is.

Open schedule for 2-node ring?

In any leader election algorithm, the two nodes must learn about each other! We stop execution when first message is received (on whatever link).

We can do this because it’s an asynchronous world (no simultaneous arrivals, delay accordingly)...

So other edge is open:

Nodes don’t know, is it an edge, or is it more?

\[ QED \]
Open Schedules for Larger Rings?

By gluing together (at the two open edges) two rings of size \( n/2 \) for which we have open schedules, an open schedule can be constructed on a ring of size \( n \). Let \( M(n/2) \) denote the number of messages used in each of these schedules by some algorithm \( \text{ALG} \). Then, in the entire ring \( 2M(n/2) + n/4 \) messages have to be exchanged to solve leader election.

Proof? Open schedule?

\[ \text{n-node Ring} \]

\[ \begin{align*}
\text{By gluing together (at the two open edges) two rings of size } n/2 \text{ for which we have open schedules, an open schedule can be constructed on a ring of size } n. \text{ Let } M(n/2) \text{ denote the number of messages used in each of these schedules by some algorithm } \text{ALG}. \text{ Then, in the entire ring } 2M(n/2) + n/4 \text{ messages have to be exchanged to solve leader election.}
\end{align*} \]
When I close one of the edges at least $n/4$ message receptions are triggered! And schedule still open. (Other edge unaffected.)

Idea: glue together at open edge; before closing an edge rings cannot distinguish whether $n/2$- or $n$ node-ring!

Assume ALG needs $M(n/2)$ messages here:
Proof of Lemma: By Induction

- Consider the ring of size \( n \) and divide it in two „subrings“ \( R_1 \) and \( R_2 \). As long as no message comes from outside, nodes cannot distinguish these two rings from two rings of size \( n/2 \). (Just delay messages accordingly: all other messages of algorithm are sent.)

- So nodes exchange \( 2 \cdot M(n/2) \) messages (induction hypothesis) in the subrings before learning anything about the other subring. Wlog assume \( R_1 \) has max ID. So each node in \( R_2 \) must learn that ID, which requires at least \( n/2 \) message receptions.

- So there must be an edge connecting the two rings that „produces“ (= triggers, but not necessarily transmits!) at least \( n/4 \) messages. Schedule/close this edge and leave other open... => open schedule for larger ring! And enough messages! 😊
How to Construct an Open Schedule?

Take-Away

Just let asynchronous algorithm run and stop before last edge is closed (i.e., before message arrives).

Why $\geq n/4$ messages triggered by border edge even if schedule is made open?

1. Maybe this is whole ring: so much information must be transferred eventually!

2. Fact independent of schedule: learning about events / timing of other edges requires $n/4$ messages at least as well!
Open Schedules for Larger Rings?

**Theorem**

Any algo needs at least $\Omega(n \log n)$ messages.

**Proof by induction:** Claim follows from maths...

\[
M(n) = 2 \cdot M \left( \frac{n}{2} \right) + \frac{n}{4}
\geq 2 \cdot \left( \frac{n}{8} \left( \log \frac{n}{2} + 1 \right) \right) + \frac{n}{4}
= \frac{n}{4} \log n + \frac{n}{4} = \frac{n}{4} (\log n + 1)
\]

So we are optimal.

Can we do better? 😊
Take-Away

In synchronous systems, not receiving a message is also information!

Idea for message complexity n? E.g., find minimum ID in environment where nodes have unique but arbitrary integer IDs (but n known)...

Sync Leader Election

- each node v does the following:
  - Divide time into phases of n steps (leaves time for lower-ID nodes to broadcast...)
  - If phase = v and did not get a message:
    - v becomes leader
    - v sends „I am leader!“ to everybody!

Breaks message lower bound but we may wait long!
Runtime O(n*minID)? What is the time – message tradeoff?
Literature for further reading:

- Attiya/Welch (Alg. 3.1 for example)
- Peleg‘s book

End of lecture