Advanced Network Security

2. Distributed Algorithms: Leader Election

Jaap-Henk Hoepman

Digital Security (DS)
Radboud University Nijmegen, the Netherlands
@xotoxot // ⬇️ jhh@cs.ru.nl // 🌐 www.cs.ru.nl/~jhh
Leader election: motivation

- **IBM token ring (1970)**
  - For local area network
  - Single token traversing ring
  - Station with token was allowed to send

- **How to**
  - Start the network?
    - $0$ tokens
  - Recover from an error?
    - $>1$ tokens
Leader election (1)

- **Given a graph** \( G = (V, E) \) **of nodes**, design a protocol that will elect a single node as leader

- **Output stored in local variable** \( C[i].leader \)
  - There is one node \( i \) with \( C[i].leader = true \)
  - For all \( j \in V, j \neq i \) we have \( C[i].leader = false \)

- **Assumptions**
  - \( G \) is connected, i.e. nodes can reach each other; we assume a bidirectional ring network here
  - Nodes have unique identifiers \( C[i].id \)
    - *E.g. a MAC address*
    - *Note that nodes do NOT know V (i.e. the set of identities in the graph)*
Leader election (2)

- **Requirements**
  - Correctness: at most one leader is elected (and once elected stays elected).
  - Progress: eventually a leader is elected.

- **Leader election used, for example to**
  - Recover from errors (the leader coordinates the repair)
  - Initiate another higher-level distributed algorithm
How would/could you solve this?
Some non-solutions

- **Node $i$ with $C[i].id = 0$ becomes leader**
  - May not exist. E.g. if identifiers are based on MAC addresses

- **Consider the following protocol for node $i$**

  Send clockwise (right) $C[i].id$
  Receive counterclockwise (left) $id$
  $C[i].leader = (C[i].id < id)$

- This protocol assumes that $C[i].id = i$, i.e. assigned increasing along the ring, this is not necessarily the case
LeLann’s protocol: leader election on a ring

Assumption
- FIFO message passing and unique identifiers
- Note: nodes do not know the size of the ring
- Unidirectional communication (clockwise only)

Protocol for node $i$

\[
\begin{align*}
I_i &= \emptyset \\
C[i].leader &= false \\
send right C[i].id \\
while \ C[i].id \not\in I_i \\
do receive left id \\
I_i &= I_i \cup \{id\} \\
send right id \\
C[i].leader &= (C[i].id = \min_{j \in I_i})
\end{align*}
\]

Election point
LeLann

- Why does this work?

- What is the message / round complexity?

- What if message passing is not FIFO?

---

\[
I = \emptyset \\
C[i].leader = false \\
\text{send right } C[i].id \\
\text{while } C[i].id \notin I \\
\text{do receive left } id \\
\quad I = I \cup \{id\} \\
\quad \text{send right } id \\
C[i].leader = (C[i].id = \min j \in I)
\]
\[ I = \emptyset \]
\[ C[i].leader = \text{false} \]
\[ \text{send right } C[i].id \]
\[ \text{while } C[i].id \notin I \]
\[ \text{do receive left } id \]
\[ \quad I = I \cup \{id\} \]
\[ \quad \text{send right } id \]
\[ C[i].leader = (C[i].id = \min j \in I) \]
LeLann

- **Why does this work?**
  - See proof further on

- **What is the message / round complexity?**
  - Every node forwards messages until it receives its own id back.
  - If the size of the ring is $n$, each node sends $n + 1$ messages
  - Total number of messages sent it $n(n + 1)$
  - Round complexity is $n$

- **What if message passing is not FIFO?**
  - Homework ;-)
LeLann proof of correctness (1)

Correctness: at most one leader is elected (and once elected stays elected).

- We need to prove that for all nodes $i,j$ that reach the election point, we have $I_i = I_j$. Then the result follows as nodes have unique identifiers.

- In fact we will show that we have $I_i = I_j = \{C[k].id|k \in [0,n-1]\}$ (nodes are numbered clockwise around the ring, and $n$ is the number of nodes - which is unknown to the number of nodes!)

- In what follows, let $I_i$ be the **list** of values, in the order in which they were received (instead of a set).
LeLann proof of correctness (1)

- [correctness proof continued]
  - We prove this using induction on the $r$-th message node $i$ receives; in fact we show that when node $i$ receives the $r$-th message, it actually received $I_i = (C[i - 1 \mod n].id, ..., C[i - r \mod n].id)$ in that order.
  - For round $r = 0$ (i.e. initially) the statement holds trivially: no messages have been received so far and $I_i = ()$
LeLann proof of correctness (2)

- [correctness proof continued]
  - For round $r' = r + 1$ observe
    - All message received by node $i$ by round $r' = r + 1$ must have been sent by its left hand neighbour $j = i - 1 \mod n$ in or before round $r$.
    - At the end of round $r$ node $j$ has sent all values in $l_j$ to $i$, in the same order, but first sent out $C[j].id$
    - Because of the FIFO property node $i$ receives these in the same order
    - Using the induction hypothesis $l_j = C[j - 1 \mod n].id, ..., C[j - r \mod n]$ in that order.
    - Then $i$ receives $C[j].id, ..., C[j - r \mod n] = C[i - 1 \mod n].id, ..., C[i - r'].id$ in that order as required.
LeLann proof of correctness (3)

[correctness proof continued]

- If node $i$ reaches the election point then $C[i].id \in l_i$
- This happens when the $r$-th message node $i$ receives (so $r > 0$), i.e. the message $C[k].id$ with $k = i - r \mod n$, equals $C[i].id$.
- As identities on the ring are unique, this implies $k = i$ and so $r = n$ and hence, $l_i = \{C[i - 1 \mod n].id, ..., C[i - n \mod n].id\}$
- In other words $l_i = \{C[k].id | k \in 0, n - 1\}$ as required.
LeLann proof of correctness (3)

**Progress: eventually a leader is elected.**

- Let node $i$ have the smallest $C[i].id$
- Initially node $i$ sends $C[i].id$ to its right-hand neighbour
- This means a message $C[i].id$ is either in transit on a link (meaning the next node will eventually receive it) or received by the node (meaning it will be sent out to the right by that node)
- Whenever this message is sent, it moves one step closer back to node $i$
- Eventually node $i$ receives $C[i].id$ (and sends it once more the right) and then stops
- It determines that $(C[i].id = \min_{i \in I} i)$ and hence becomes leader as required
What if nodes do not have unique identifiers?

deterministic

same state initially
What if nodes do not have unique identifiers?

- Then there exists a symmetric configuration $C$
  - where all nodes have the same state, and all edges have the same state
  - I.e. either all nodes are leaders, or no node is leader

- Starting in $C$ let all nodes take a step (the same) in turn, then
  - all steps are local steps (changing the local state to a new state, the same for all nodes)
  - all steps are receive actions (receiving the same message), or
  - all steps are send actions (sending the same message)

- Therefore the resulting configuration $C'$ is again symmetric

- We can repeat this forever, never reaching a state where there is exactly one leader

- This is called a symmetry argument
Peterson’s protocol: leader election on a ring

- **Bidirectional communication**
  - Nodes can send messages clockwise and anticlockwise

- **Idea: algorithm proceeds in rounds**
  - First round all $n$ nodes are *active* and participate
  - If a round starts with $k$ participants, at least $k/2$ and at most $k - 1$ will be eliminated (and become *passive*)
  - If a round start with 1 participant, it will declare itself leader at the end of the round
Peterson’s protocol: leader election on a ring

The protocol for node $i$

$C[i].active = true$
$C[i].leader = false$

while true /* new round */
do if $C[i].active == true \land C[i].leader == false$
    send left $C[i].id$
    send right $C[i].id$
    receive right rightid
    receive left leftid
    if $(C[i].id == leftid) \land (C[i].id == rightid)$
        $C[i].leader = true$
    else if $(C[i].id < leftid) \lor (C[i].id < rightid)$
        $C[i].active = false$
else /* passive or leader */
    receive right id ; send left id
    receive left id ; send right id
Peterson’s protocol

- Why does it work?

- What is the message / round complexity?

- What if message passing is not FIFO?

```plaintext
C[i].active = true
C[i].leader = false
while true /* new round */
  do if C[i].active == true ∧ C[i].leader == false
      send left C[i].id
      send right C[i].id
      receive right rightid
      receive left leftid
      if (C[i].id == leftid) ∧ (C[i].id == rightid)
        C[i].leader = true
      else if (C[i].id < leftid) ∨ (C[i].id < rightid)
        C[i].active = false
      else /* passive or leader */
        receive right id; send left id
        receive left id; send right id
```


```plaintext
C[i].active = true
C[i].leader = false

while true /* new round */
    do if C[i].active == true ∧ C[i].leader == false
        send left C[i].id
        send right C[i].id
        receive right rightid
        receive left leftid
        if (C[i].id == leftid) ∧ (C[i].id == rightid)
            C[i].leader = true
        else if (C[i].id < leftid) ∨ (C[i].id < rightid)
            C[i].active = false
        else /* passive or leader */
            receive right id ; send left id
            receive left id ; send right id
```

---

**Peterson**
Peterson’s protocol: leader election on a ring

- **There are at most** $\log n$ **rounds**
  - Node can only survive (remain active) if both its left and right active neighbour are smaller
  - Therefore at most half of the nodes can survive

- **In every round** $2n$ **messages are sent**
  - An active or passive node sends exactly 2 messages in each round

- **So: message complexity is at most** $2n \log n$

---

```plaintext
C[i].active = true
C[i].leader = false
while true /* new round */
do if C[i].active == true ∧ C[i].leader == false
  send left C[i].id
  send right C[i].id
  receive right rightid
  receive left leftid
  if (C[i].id == leftid) ∧ (C[i].id == rightid)
    C[i].leader = true
  else if (C[i].id < leftid) ∨ (C[i].id < rightid)
    C[i].active = false
  else /* passive or leader */
    receive right id; send left id
    receive left id; send right id
```