CSc72010

Leader Election

Leader election algorithms

The basic solution to cycles in an Ethernet LAN is for the switches to run the spanning tree protocol (STP), which determines a spanning tree for the LAN. The switches then forwards frames only through ports that are part of a spanning tree.

Introduction

Model:

Network of identical processes
Algorithm to pick a leader
Motivation: see above; token ring; commit coordinator; resource allocator

Requirement: Sometime after we start the algorithm, exactly one process outputs "leader."

Folk theorem: cannot use identical processes
Proof: By induction on number of rounds
Idea: If all processes are in the same state at round r, they must be in the same state at round r+1 (otherwise, they are not identical).
Therefore, if any process declares itself leader, they all do

Can we get away with just an ID distinguishing the processes? Each process knows its own UID and can distinguish its neighbors.

Leader election in a ring

Special case: Network is ring – still has some of the key difficulties

```
We draw a ring 1->2->3->4
               ^        |
               |------------
```

but the processes don't know the numbers; all they can do is distinguish clockwise and counter-clockwise neighbors

Lelann-Chan-Roberts (LCR)

Assume unidirectional ring (note that for learning bridges, we must have bidirectional).
Processes don't know network size
Processes can tell if two UID's are the same or different; no arithmetic allowed.

Informally: each process sends its own identifier to its clockwise neighbor. A process forwards a received UID if it is larger than its own; drops it if smaller; declares itself leader if it receives its own UID

IOA
M = {UID}
Let U designate the type of the UID's

```
states
        u: U                    % the UID of the process
        send: M, initially null
        status: enumeration of unknown, leader, initially unknown
msgs
        send the message in send to the next process clockwise
trans
        send := null
        if incoming = v then
                case
                v>u: send := v
                v=u: status := leader
                v<u: % do nothing
                end case
        end if
```

Correctness
Eventually, one (and only one) node outputs "leader"

Let n be the size of the ring (number of processes)
Define i_{max} to be the number of the process with the maximum UID
Define u_{max} to be the maximum UID

Prove:
1) i_{max} outputs leader by the end of round n
2) No other process outputs leader

Assume nodes are numbered 0,…,n-1

Lemma For 0 <= r <= n-1, after r rounds, $send_{i_{max}+r} = u_{max}$, where $i_{max}+r$ is computed using arithmetic modulus n.
Proof: By induction on r. The idea is that if a $send_i$ contains the largest UID at the beginning of a round, then $send_{i+1}$ will contain it at the end of the round.

The special case r=n-1 establishes 1) above.

For 2)
Lemma. For any r>=0, after r rounds, if $i != i_{max}$ and j is in $[i_{max}, i)$ then $send_j != u_i$.
The idea is that no UID gets past i_{max}, which contains the largest UID.
Proof is by induction on r.

Note that we are proving invariants in the lemmas – properties that are true in all reachable states.
Standard proof technique:

1) Define an invariant
2) Prove by induction. The induction step examines case-by-case what the transition function does.

Complexity
Communication (how many messages total): n^2 messages
Time (how many rounds): n

Variants

Processes stop when the leader has been output:
Leader must circulate new message

Non-leader announcements
When a process receives a larger UID

Reduced complexity
Can we reduce messages below n^2?
Yes: Hirschberg-Sinclair

Assumptions
Bidirectional
Don't know ring size
Comparisons only for UID's

Informally: Send the UID both directions to successively greater distances (double the distance on each pass.
Outbound: Pass on the UID if it's larger than your own, swallow it otherwise.
Inbound: Pass everything on.
If you get your own UID inbound, proceed to the next round
Details on page 33 of text.

Number of rounds is log n (until doubling gets us to the size of the ring).
Number of messages in a round is O(n).

Invariants:
For every phase k, 0 <=k<=1+floor(log n), and every round r, 0<=r<2^k,
$send^{+}_{i_{max}+r} = u_{max}$ and for every round r 2^k<=r<=2^(k+1), $send^{-}_{i_{max}+2^{k+1}=r}=u_{max}$.

Can verify that u_{max} gets back to the originator and therefore survives each round.

Details next time