Leader Election

Goal of leader election

$G = (V, E)$ defines the network topology.

Each process $i$ has a variable $L(i)$ that defines the leader.

- $i,j \in V : i,j$ are non-faulty ::
- $L(i) \notin V$
- $L(i) = L(j) \land L(i)$ is non-faulty

Often reduces to a maxima (or minima) finding problem.
Bully Algorithm

**Step 1.** Send an *election* message to every process with a higher identifier. *(I want to be the new leader)*

**Step 2.** Give up if any process with higher id responds. Wait to receive a *leader* message *(I am the leader)* from some process with a higher id.

**Step 3.** If no process with a higher id responds, then elect yourself as the leader, and send a *leader* message to all.

**Step 4.** If you don’t receive a *leader* message after receiving a response to your *election* message, perhaps the winner of the election failed, so reinitiate the election.
Maxima Finding

Chang-Roberts algorithm on unidirectional ring

Uses token circulation. Each token contains a process id. Processes have colors red or black. Initially all are red (i.e. they are candidates)

{For each initiator: candidates are red}

\[
\text{do } \text{token } <j> \text{ if } j < i \text{ then skip \{j’s token is removed by i\}} \\
\text{token } <j> \text{ if } j > i \text{ then send } <j>; \text{ color := black \{i resigns\}} \\
\text{token } <j> \text{ if } j = i \text{ then } L(i) := i \text{ \{i becomes the leader\}} \\
\text{od}
\]

{For a non-initiator process: either black or will turn black}

\[
\text{do } \text{token } <j> \text{ received \{color := black; send } <j> \text{ od}
\]

What is the worst-case message complexity?
Franklin’s algorithm on bidirectional ring

Similar ideas. Exchange tokens in rounds. In each round some red process turns black.

\[
\{\text{program for a red process } i \text{ in round } r, r \geq 0\}
\]

send token \(<i>\) to both neighbors;
receive tokens from both neighbors;

if token \(<j> : j > i \Rightarrow \quad \text{color := black}

\text{token } <j> : j < i \Rightarrow \quad r := r+1; \text{ go to the next round}

\text{token } <j> : j = i \Rightarrow \quad L(i) := i

fi

Complexity. \(\log_2 n\) rounds and \(O(n)\) messages per round.
Faults and fault-tolerance

Various views of faults
   Cause or effect? Both views seem to be well accepted.

Some well-known classes
   Crash failure
   Omission failure
   Transient failure
   Byzantine failure
   Fail-stop failure
   Software failure
   Temporal failure

What are these?
Specification of faults

Example 1.

```
program example1;
define x : boolean, message : a or b;
initially x = true;
do x <> send a  \{specified actions S\}
  true <> send b  \{fault actions F\}
od
```

What if \( F \equiv true \Rightarrow x := false? \)
Example 2. A process receives a message and forwards it to each of its \( N \) neighbors \( 0, 1, 2, \ldots, N-1 \). The fault-free system can be specified by

```plaintext
program example 2;
define j: integer, flag: boolean, message : a or b
initially j = 0, flag = false

{Specified actions S}
do ¬flag " message = a " x := a; flag := true
   (j < N) " flag " send x to j; j := j+1
   j = N " j := 0; flag := false
od

Now, add the following fault action:

F: flag " x:= b {b ≠ a}

It models a class of byzantine failures.
Temporal failure and timeout

program for process j;
define f[i]: boolean  {initially f[i] = false}

S:  do ¬ f[i] □ message from process i □ skip
F:  timeout (i,j) □ f[i]:= true
   od

Correct use of timeout requires synchronized clocks.
Types of tolerance

<table>
<thead>
<tr>
<th>Crash</th>
<th>Byzantine</th>
</tr>
</thead>
<tbody>
<tr>
<td>masking</td>
<td></td>
</tr>
<tr>
<td>non-masking</td>
<td></td>
</tr>
<tr>
<td>graceful degradation</td>
<td></td>
</tr>
<tr>
<td>fail-safe</td>
<td></td>
</tr>
</tbody>
</table>

Masking tolerance
The application should always behave as expected.
Examples?

Non-masking tolerance
Temporary glitches are ok, but the system should eventually recover and behave normally. Examples?

fault occurs here

recovery

fault occurs here

recovery

recovery completes here

recovery completes here
Backward error recovery

Forward error recovery

Graceful degradation

Fail-safe systems