CS 138: Self-Stabilizing Systems
Token Ring
Token Ring Problem (1)
Token Ring Problem (2)
Enter Dijkstra
Self-Stabilizing Systems

- A distributed system has a set of legal states
- Suppose it’s zapped by some outside force and enters an illegal state

- Can it be constructed so that it is guaranteed to return to a legal state in a bounded amount of time?
Notation, etc.

- **Guarded commands**
  
  \[ \text{guard} \rightarrow \text{command} \]
  
  - execute \textit{command} when \textit{guard} is true

- **Token ring**
  
  - \texttt{node.state}
    
    - integer state of node
  
  - \texttt{node.next}
    
    - next node (clockwise)
  
  - \texttt{node.prev}
    
    - previous node (counter clockwise)
Solution

• N nodes, each with k states, $k > N$
• Special distinguished node
  \[
  (\text{node.prev.state} == \text{node.state}) \rightarrow \\
  \text{node.state}++(\text{mod } k)
  \]
• All other nodes
  \[
  (\text{node.prev.state} != \text{node.state}) \rightarrow \\
  \text{node.state} = \text{node.prev.state}
  \]
• Legal system states
  – exactly one guard is true
Example (1)
Example (2)
Example (3)
Example (4)
Example (5)
Example (6)
Example (7)
Example (9)
Example (10)
Example (12)
Example (13)
Example (14)
Also …

• Gave solutions with 4-state machines and 3-state machines
• Someone later proved that it cannot be done with 2-state machines
Proof

• Dijsktra didn’t bother …
• It’s up to us
Proof (1)

• Explain why it is that at any particular moment, at least one guard must be true, even if the system has been zapped

• Special distinguished node
  
  (node.prev.state == node.state) →
  
  node.state++(mod k)

• All other nodes
  
  (node.prev.state != node.state) →
  
  node.state = node.prev.state
Proof (2)

• Show that if all nodes have the same value for their states, the system is stable
  – stable: the system is in a state in which only one node’s guard is true; whenever the system changes global state legally, it goes to a global state in which the next node’s guard is the only one that’s true

• Special distinguished node

  $$(\text{node.prev.state} == \text{node.state}) \rightarrow \text{node.state}++(\text{mod } k)$$

• All other nodes

  $$(\text{node.prev.state} != \text{node.state}) \rightarrow \text{node.state} = \text{node.prev.state}$$
Proof (3)

• Show that if node 0’s state is greater than those of all other nodes, the system will necessarily reach a stable global state.

• Special distinguished node
  \[(\text{node.prev.state} == \text{node.state}) \rightarrow \text{node.state}++(\text{mod } k)\]

• All other nodes
  \[(\text{node.prev.state} != \text{node.state}) \rightarrow \text{node.state} = \text{node.prev.state}\]
Proof (4)

• Assume now that each node’s state value is an unbounded non-negative integer (i.e., k is infinite). Show that, regardless of its current state, the system will necessarily reach a global state in which node 0’s state is greater than those of all others.

• Special distinguished node
  \[(\text{node.prev.state} == \text{node.state}) \rightarrow \text{node.state++(mod k)}\]

• All other nodes
  \[(\text{node.prev.state} != \text{node.state}) \rightarrow \text{node.state} = \text{node.prev.state}\]
Proof (5)

• Redo part 4, this time assuming $k \geq n$: the system will necessarily reach a global state in which node 0’s state is greater than those of all others

  • Special distinguished node
    $\text{(node.prev.state == node.state)} \Rightarrow$
    $\text{node.state++ (mod k)}$

  • All other nodes
    $\text{(node.prev.state != node.state)} \Rightarrow$
    $\text{node.state = node.prev.state}$
Proof (6)

• Show that the system won’t necessarily ever enter a stable state after being zapped if $k < n$

• Special distinguished node
  \[(\text{node.prev.state} == \text{node.state}) \rightarrow \text{node.state}++(\text{mod } k)\]

• All other nodes
  \[(\text{node.prev.state} != \text{node.state}) \rightarrow \text{node.state} = \text{node.prev.state}\]
Another Problem ...

Find a spanning tree
Easier...

One node is special.
Algorithm (1)

- Node’s value is distance from root.
  - initialize all to 0
Algorithm (2)

• Each non-root node looks at neighbors’ values
  • sets its value to min + 1
  • sets pointer to min neighbor
Algorithm (3)

- Each non-root node looks at neighbors’ values
  - sets its value to min + 1
  - sets pointer to min neighbor
Algorithm (n)

• Each non-root node looks at neighbors’ values
  • sets its value to min + 1
  • sets pointer to min neighbor
Layered Self-Stabilization

Application reset
Layered Self-Stabilization

- A node discovers that something is dreadfully wrong
Layered Self-Stabilization

- It gets all nodes to do a reset, and thus all start in initial state.
Issues

• Without global synchronization, not all will be in initial state at once
  – settle for a state that could have been reached from such a global initial state
• How can reset be propagated in a self-stabilizing manner?
Propagation

• Basic idea
  – reset-initiator sends request through spanning tree to root
  – root sends reset request to all nodes via spanning tree
  – once leaves get reset request, they send ack back to root
Propagation

- Details (approximate …)
  - each node has application state
    - normal, initiate, reset
  - resetting node sets state to initiate
    - sends request towards root
    - intermediate nodes set states to initiate
  - root receives request, sets state to reset
    - sends request to leaves
    - each intermediate node sets state to reset
  - leaves receive request, set state normal
    - send ack back to root
    - intermediate nodes (and root) set state to normal
Propagation

• Details (more exact)
  – resetting node sets state to initiate from normal
    - sends request towards root
    - intermediate nodes in normal state set states to initiate; if not in normal state, do nothing
  – root receives request, sets state to reset if in normal state
    - sends request to leaves
    - each intermediate node sets state to reset
  – leaves receive request, set state normal
    - send ack back to root
    - intermediate nodes (and root) set state to normal if in reset state
One More Thing …

• One recently reset node may communicate with an adjacent node that has not yet reset itself
• Each node maintains session number
  – initially zero
  – incremented at each reset
  – nodes may communicate only if session numbers are equal
Probabilistic Self-Stabilization

Ethernet
Ethernet Backoff Algorithm

• Binary exponential backoff
  – after a collision, time is slotted in $2^\tau$ units
    - maximum collision-detection time: 51.2 $\mu$seconds in IEEE 802.3
  – first retry is 0 or 1 slots away (randomly selected)
  – if collide again, next retry is 0, 1, 2, or 3 slots away (randomly selected)
  – after $i$ collisions, skip a random number of slots in $[0, 2^i - 1]$
  – maximum: 1023 slots