Leader Election (contd.)

Arvind Krishnamurthy
Fall 2003

Leader Election

- Recap:
  - Impossible for anonymous rings
  - Possible for non-anonymous rings
    - For asynchronous networks:
      - Message complexity: O(n log n)
      - Time complexity: O(n)
    - For synchronous networks, fewer messages are required if you use node uid to count rounds or slow messages
- Today:
  - Simple algorithm for general topology
  - Randomized algorithm for anonymous rings
  - Optimized algorithm for general topology (under synchronous execution)

General networks

- Start DFS spanning tree algorithm from all nodes
- In addition:
  - Send node's uid along with M
  - When two DFS traversals collide, the copy with the higher uid wins
    - The winner gets a response
    - The other traversal stalls – no response is sent to the sender
  - Key fact: node sends response only after all it completes the traversal of all its neighbors

Concurrent DFS

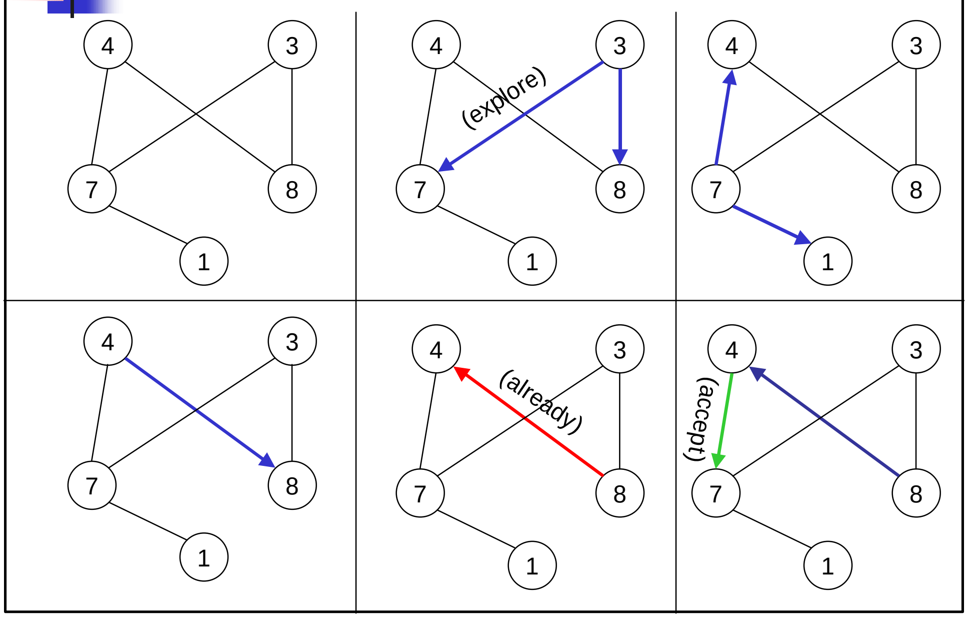
```
Initial State:
        parent = nil
        leader = 0
        neighborlist = list of adjacent nodes
        children = nil
        unexplored = neighborlist
```

```
Upon receiving no message p_i does:
        if parent == nil then
                leader = my-id
                parent = i
                let p_j be an element of unexplored
                remove p_j from unexplored
                send [ my-id ] to p_j
```

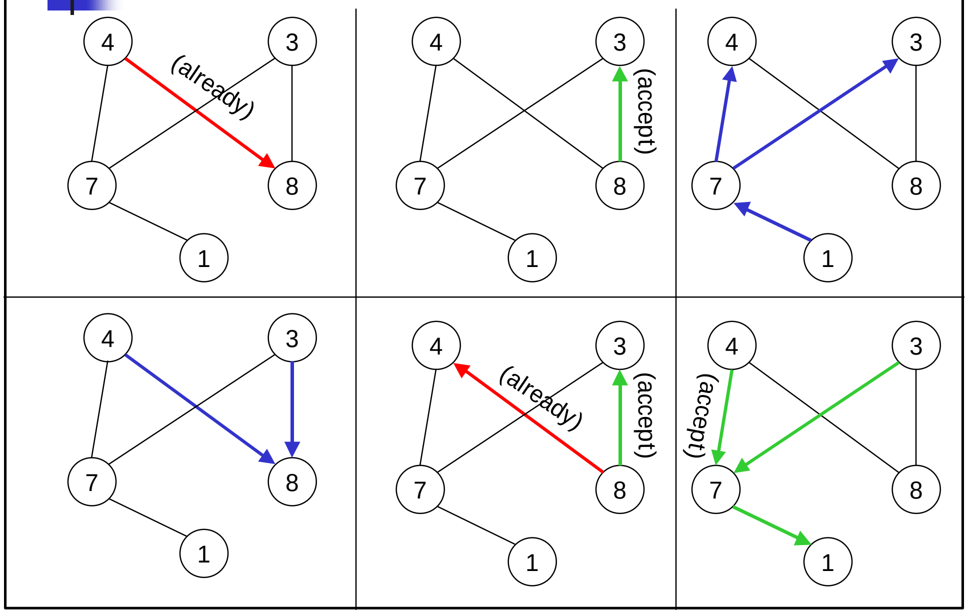
Continuing Traversal

```
Upon receiving [new-id] from neighbor p_j:
        if (leader < new-id)
                leader = new-id
                parent = j
                unexplored = neighborlist – p_j
                if unexplored != nil
                        let p_k be a processor in unexplored
                        remove p_k from unexplored
                        send [leader] to p_k
                else
                        send [accept] to parent
        else if (leader == new-id)
                send [already] to p_j
        // otherwise, do nothing
```

Sample Execution



Sample Execution



Terminating Traversal

```
Upon receiving [accept] or [already] from p_j:
        if received [accept]
                add j to children
        if unexplored == nil
                if parent != i then send [accept] to parent
                else terminate as root of the spanning tree
        else
                let p_j be an element of unexplored
                remove p_j from unexplored
                send [leader] to p_j
```

Complexity Analysis

- In the worst case:
  - There could be n concurrent traversals
  - Each traversal is O(m) messages since DFS is a flooding algorithm
  - Total number of messages = O(n m)
- Time complexity:
  - Each DFS takes O(m) time
  - DFS is performed concurrently
  - Total time complexity = O(m)

Randomized Leader Election

- Extend transition function to accept as input:
  - A random number
  - From a bounded range
  - Under some fixed distribution
  - Used once or some number of times
- The bad news:
  - Randomization alone does not generally affect:
    - Impossibility results: leader election in anonymous networks is still impossible!
    - Worst case bounds
- The good news: randomization + weakening of problem statement does help

Randomized Leader Election

- Elect a leader with some probability
- Weaken leader election as follows:
  - Safety: in every configuration of every admissible execution, at most one processor is in an elected state
  - Liveness: one processor is elected with some non-zero probability
- Behaviors allowed by weakened specification:
  - Terminate without a leader
  - Never terminate

Randomization

- Use randomization to have processes generate a pseudo identifier
- Use a deterministic leader election algorithm to work with these pseudo identifiers
- Not just any deterministic leader election algorithm:
  - Needs to work correctly if multiple processes generate same pseudo-id
  - Ability to detect if no leader is elected
- Consider:
  - A synchronous ring
  - Non-uniform (nodes know the value of “n”)
  - Use randomization to generate one random number

Algorithm

```
Initially:
        my-uid = 1 with probability 1 – 1/n
                 2 with probability 1/n
        send [my-uid] to left
```

```
Upon receiving M from right:
        if size of M == n then
                if my-uid == unique maximum of M then
                        elected = true
                else
                        elected = false
        else
                send [M || my-uid] to left
```

Analysis

- What is the probability that the algorithm terminates with a leader?
- What is the message complexity?

Repeated Leader Election

- Trade off more time and messages for higher probability of success
  - If size of M == n and processor detects no single maximum in M
    - Choose new uid
    - Restart algorithm
  - Random number generator is used multiple times
  - Keep repeating till you eventually succeed

- Analysis:
  - What is the probability that there is no leader elected after k rounds?
  - What is the expected case behavior of this algorithm?
    - Each iteration is an independent iteration capable of succeeding with some probability; model it as a geometric sequence

Loose Ends and Summary

- There is no uniform randomized algorithm for leader election in a synchronous anonymous ring

- Summary:
  - No deterministic solution for anonymous rings
  - No solution for uniform anonymous rings (even with randomization)
  - Protocols for O(n^2) and O(n logn) messages for uniform rings which are non-anonymous
  - Lower bound on messages for asynchronous networks: n log n
  - O(n) message complexity for uniform synchronous rings if uids can be manipulated with arbitrary operations

Announcements

- Design document:
  - Email to me
  - Text, ps, pdf documents are fine
- Assignment:
  - Build from basic blocks
  - Get a simple file-get operation to work
  - Get multithreading to work for a simple file-get
  - Add more protocol complexity in incremental fashion
  - Check for error conditions
- Design reviews tomorrow/friday

Faster Leader Election in General Networks

- General approach:
  - Build a spanning tree of the entire network
    - Each node determines a parent
  - Root of the tree is the leader
- In fact, compute not just any spanning tree:
  - Compute the “minimum spanning tree” in the network
  - Assumes that channels have some kind of “weight” or “cost” that needs to be minimized
  - Useful for determining an “efficient” subgraph over which communication can take place

Basic facts of MST

- Let T be a portion of the MST
- Find some edge:
  - That is not included in T
  - Which does not create a cycle when added to T
  - Has the minimum weight
- Then this edge can be added to T to extend T
- Alternately:
  - Consider some connected component that belongs to MST
  - Consider the minimum-weight outgoing edge (MWOE) from that component
    - Outgoing implies that edge does not create a cycle (nor is it currently included in the component)
  - This edge can be included to extend the connected component
- Prim-Dijkstra: start with one vertex and build MST
- Kruskal: start with “n” components and combine them with MWOE

Can we build a concurrent version of Kruskal’s algorithm?

- General idea:
  - Each component finds its MWOE
  - The MWOEs are added concurrently
  - Unfortunately, it might create cycles!
- Solution:
  - Assume that edge weights are unique
  - Can generate unique edge weights by combining processor uids into the edge weight