Distributed Computation

- How to send messages to all nodes efficiently
- How to compute sums of values at all nodes efficiently
- Broadcasting messages
- Computing sums in a tree
- Computing trees in a network
Network as a graph

• Diameter
  – The maximum distance between 2 nodes in the network

• Radius
  – Half the diameter

• Spanning tree of a graph:
  – A subgraph which is a tree, and reaches all nodes of the graph
  – If network has n nodes
    • How many edges does a spanning tree have?
Computing sums in a tree

- Suppose root wants to know sum of values at all nodes
Computing sums in a tree

- Suppose root wants to know sum of values at all nodes
- It sends “compute” message to all children
- They forward the message to all their children (unless it is a leaf node)
- The values move upward from leaves
- Each node adds values from all children and its own value
- Sends it to its parent
Computing sums in a tree

- What can you compute other than sums?
- How many messages does it take?
- How much time does it take?
Global Message broadcast

• Message must reach *all nodes in the network*
  – Different from broadcast transmission in LAN
  – All nodes in a large network cannot be reached with single transmission
Global Message broadcast

• Message must reach *all nodes in the network*
  – Different from broadcast transmission in LAN
  – All nodes in a large network cannot be reached with single transmissions
Flooding for Broadcast

• The source sends a *Flood* message to all neighbors

• The message has
  – Type *Flood*
  – *Unique id*: (source id, message seq)
  – *Data*
Flooding for Broadcast

• The source sends a *Flood* message, with a unique message id to all neighbors

• Every node p that receives a flood message m, does the following:
  – *If m.id was seen before, discard m*
  – *Otherwise, Add m.id to list of previously seen messages and send m to all neighbors of p*
Flooding for broadcast

• Storage
  – Each node needs to store a list of flood ids seen before
  – If a protocol requires $x$ floods, then each node must store $x$ ids
    • (there is a way to reduce this. Think!)
Assumptions

• We are assuming:
  – Nodes are working in synchronous communication rounds (e.g. transmissions occur in intervals of 1 second exactly)
  – Messages from all neighbors arrive at the same time, and processed together
  – In each round, each node can successfully send 1 message to each neighbor
  – Any necessary computation can be completed before the next round
Communication complexity

• The message/communication complexity is:
Communication complexity

• The message/communication complexity is:
  – \( O(|E|) \)
Communication complexity

• The message/communication complexity is:
  – $O(|E|)$
  – Worst case: $O(n^2)$
Reducing Communication complexity (slightly)

- Node $p$ need not send message $m$ to any node from which it has already received $m$
  - Needs to keep track of which nodes have sent the message
  - Saves some messages
  - Does not change asymptotic complexity
Time complexity

• The number of rounds needed to reach all nodes: \textit{diameter of } G
Computing Tree from a network

• BFS tree
  – The Breadth first search tree
  – With a specified root node
BFS Tree

• Breadth first search tree
  – Every node has a *parent* pointer
  – And zero or more child pointers

  – BFS Tree construction algorithm sets these pointers
BFS Tree Construction algorithm

- Breadth first search tree
  - The \textit{root(source)} node decides to construct a tree
  - Uses flooding to construct a tree
  - Every node \( p \) on getting the message forwards to all neighbors
  - Additionally, every node \( p \) stores \textit{parent} pointer: node from which it first received the message
    - If multiple neighbors had first sent \( p \) the message in the same round, choose \textit{parent} arbitrarily. E.g. node with smallest id
  - \( p \) informs its parent of the selection
    - Parent creates a child pointer to \( p \)
BFS Tree

• Property: BFS tree is a shortest path tree
  – For source s and any node p
  – The shortest path between s and p is contained in the BFS tree
Time & message complexity

• Asymptotically Same as Flooding
Tree based broadcast

• Send message to all nodes using tree
  – BFS tree is a *spanning* tree: connects all nodes

• Flooding on the tree

• Receive message from parent, send to children
Tree based broadcast

• Simpler than flooding: send message to all children

• Communication: Number of edges in spanning tree: n-1
Aggregation: Find the sum of values at all nodes

- With BFS tree
  - Start from *leaf* nodes
    - Nodes without children
    - Send the value to parent
  - Every other node:
    - Wait for all children to report
    - Sum values from children + own value
    - Send to parent
Aggregation

• Without the tree

• Flood from all nodes:
  – \(O(|E|)\) cost per node
  – \(O(n^* |E|)\) total cost: expensive
  – Each node needs to store flood ids from \(n\) nodes
    • Requires \(\Omega(n)\) storage at each node

• Good fault tolerance
  • If a few nodes fail during operation, all the rest still get some value
Aggregation

• With Tree

• Also called Convergecast
Aggregation

• With Tree

• Once tree is built, any node can use for broadcast
  – Just flood on the tree

• Any node can use for convergecast
  – First flood a message on the tree requesting data
  – Nodes store parent pointer
  – Then receive data

• What is the drawback of tree based aggregation?
Aggregation

• With Tree

• Once tree is built, any node can use for broadcast
  – Just flood on the tree

• Any node can use for convergecast
  – First flood a message on the tree requesting data
  – Nodes store parent pointer
  – Then receive data

• Fault tolerance not very good
  – If a node fails, the messages in its subtree will be lost
  – Will need to rebuild the tree for future operations
Computing Trees:

• What if the edges have weights?
Aggregation using Trees:

- What if the edges have weights?
- The cost may not be $O(n)$ since weights can be higher

- How to get the best performance?
Minimum spanning tree is

• A spanning tree (reaches all nodes)
• With minimum possible total weight

• How can we compute a minimum spanning tree efficiently in a distributed system?
• (remember, a node knows only its neighbors and edge weights)