Broadcasting (cont'd)

References for today: [AW98], Sections 2.4, 2.5. More on this topic in [Lyn96], SEctions 15.3-15.5.

Last Day: We saw that flooding can be used to broadcast using $\Theta(m)$ messages. During that broadcast we can compute a spanning tree, which allows subsequent broadcasts to be done using $\Theta(n)$ messages (assuming no failures).

Can we do better on the first broadcast (without failure) ?
Suppose each process knows id of each of its neighbours.
Idea:

- Do a DFS of graph to distribute the message
- Include in the message list of all procs who received message
- Then don't send message across edge if neighbour already received it

Problem: header gets really long $\longrightarrow$ this make it impractical.
Advantage: $O(n)$ messages instead of $O(m)$.
So, Can we achieve $O(n)$ with small headers ?
Improved algo: just keep list of nodes along single path from-to current location in header (backtacking idea). $\longrightarrow$ still big headers in worst case.

Model
- asynchronous
- message-passing
- proc has no knowledge of network topology (just number of adjacent channels)
- constant size messages

Assumption If no broadcast occurs no message should be sent.

Claim Broadcast requires $\Omega(m)$ messages.

Proof Show that we will need to send at least one message across each edge.

If we don't use some edge e in a network N, we're going to change the setup of the network to N' where algo fails. Create new network N' which is the same as n, except it has one new vertex v on middle of edge e. If we consider running a broadcast in these two network N and N' the set of messages sent will be exactly the same because no node (except v) can tell the difference between the two executions α and α'.
Note: v can't send any messages in α' because v cannot tell the difference between execution α' and an execution where no broadcast happens.
In α' vertex v does not receive the broadcast message.
$\Longrightarrow$ algo is not correct
$\Longrightarrow$ no correct algo (in this model) can broadcast using less than m messages

Generalization: Broadcast requires $\Theta(min(m, n^{1+\Theta(1)/\rho}))$ messages for broadcast if you initially know processes within radius ρ. (Shown by Awerbuch, Goldreich, Peleg, Vainish.)

Spanning Tree

It would be nice to build a single spanning tree in graph as soon as anybody wants to do a broadcast.
If processes have unique ids then each processor who starts building a spanning tree can associate its own id with that tree. When a processor sees two different spanning trees constructed it can ignore the one with larger id.
Suppose everyone starts setting up a spanning tree, processors follow flooding algorithm for a message if id in header is the lowest it has seen so far otherwise it kills the message.
If q has the smallest id in the system, the tree initiated by q will eventually be built completly. No other tree will ever be completed because node q won't send any ack. q will be the only processor who gets back all acks, so it knows when the tree is complete.
messages is $O(mn)$ since n trees are initiated and each tree creates $O(m)$ messages.

ConvergeCast

Instead of sending info to everyone, one processor is gathering info from everyone.
$\Longrightarrow$ send info to the root of the tree.
Many problems in distributed computing can be thought of as computing a function of private inputs.
$\longrightarrow$ Each processor get an input v_i
$\longrightarrow$ Each processor must output $f(v_1, \ldots, v_n)$
Solution:

- Convergecast
- root computes f
- broadcast result

So solving broadcast problem allows us to solve lots of other problems.

This method for computing f takes $O(n)$ messages if a spanning tree has already been computed. (This is better than having every process broadcast its input value to every other process so that each process can compute f.)

Leader election

Ring of processors

$\longrightarrow$ most basic kind of network.
impossibility in ring $\Rightarrow$ not [possible to solve problem in every network]
$\exists$ algo that works in a ring $\Rightarrow$ $\exists$ algo that work in every network with an embedded ring (eg. hypercube).

Leader election

Every processors must output 0 or 1 and exactly one processor outputs 1. Principle application: Having a single proc that coordinates actions among other processors.