

# Broadcast

Broadcasting is an important communication mode: sending a message from a source node to all other nodes of the network.

Two basic broadcasting approaches: flooding and spanning tree-based.

## **Flooding:**

A source node  $s$  wants to send a message to all nodes in the network.

$s$  simply forwards the message over all its edges.

Any vertex  $v \neq s$ , upon receiving the message for the *first* time (over an edge  $e$ ) forwards it on every other edge.

Upon receiving the message again it does nothing.

# Analysis

**Theorem 1.** *The message complexity of flooding is  $\Theta(|E|)$  and the time complexity is  $\Theta(D)$ , where  $D$  is the diameter of  $G$ .*

**Proof.** The message complexity follows from the fact that each edge delivers the message at least once and at most twice (one in each direction). To show the time complexity, we use induction on  $t$  to show that after  $t$  time units, the message has already reached every vertex at a distance of  $t$  or less from the source.  $\square$

# Tree Broadcast

Assume that a spanning tree has been constructed.  
Then:

**Theorem 2.** *For every  $n$ -vertex graph  $G$  with a spanning tree  $T$  rooted at  $r_0$ , the message complexity of broadcast is  $n - 1$  and time complexity is  $\text{depth}(T)$ .*

A broadcast algorithm can be used to construct a spanning tree in  $G$ .

The message complexity of broadcast is asymptotically equivalent to the message complexity of spanning tree construction.

Using a breadth-first spanning tree, we get the optimal message and time complexities for broadcast.

# Asynchronous Model

In an asynchronous model, no assumptions are made about any internal clocks or on the speeds of the processors.

The steps in an asynchronous algorithms are determined by conditions or *events* and not by clock ticks.

We assume that messages arrive in the same order they are sent (i.e., there is FIFO queuing).

We assume that if a processor has an event enabling it to perform a task, the processor will eventually perform the task. In particular, each message sent will eventually be delivered.

**Asynchronous time complexity:** We assume that each message incurs a time complexity of at most one time unit.

*Partially synchronous model:* Processors have some partial information about timing, e.g., almost synchronized clocks or approximate bounds on message delivery time etc.

Using a tool called *synchronizers* one can transform a synchronous algorithm to work in an asynchronous model with no increase in time complexity and at the cost of some increase in the message complexity. (Peleg's book)

# BFS tree construction

In the synchronous model, the flooding algorithm generates a BFS tree.

We focus on asynchronous model for the next two algorithms.

## Distributed Dijkstra's Algorithm

The root  $r_0$  initiates a pulse message.

The algorithm proceeds in phases. A phase consists of the following:

(Assume that the first  $p$  phases are done and the algorithm has constructed a tree  $T_p$  which is a BFS tree spanning  $\Gamma_p(r_0)$ , the  $p$ -neighborhood of  $r_0$ .)

1. The root broadcasts a *pulse* message on  $T_p$ .
2. When a leaf of  $T_p$  receives a pulse message, it sends a *layer* message to all its neighbors except its parent.

3. A vertex receiving the *layer* message for the first time from a node  $v$  makes  $v$  its parent and acknowledges it. All subsequent layer messages are acknowledged negatively.

4. Each leaf collects the acks of its layer messages.

5. Acks are convergecast on  $T_p$  to the root.

6. Once the convergecast terminates, it initiates a new phase.

Time complexity of phase  $p = 2p + 2$ .

Hence total time =  $\sum_p 2p + 2 = O(D^2)$ .

Message complexity of a phase =  $O(n) + O(E_p)$  where  $E_p$  is the set of edges between the leaves and their children.

Hence total messages =  $\sum_p O(n) + O(E_p) = O(nD + |E|)$ .

# Distributed Bellman-Ford

1. Initially, the root sets  $L(r_0) = 0$  and all other vertices set  $L(v) = \infty$ .

2. The root sends out the message  $Layer(0)$  to all its neighbors.

3. A vertex  $v$ , which gets a  $Layer(d)$  message from a neighbor  $w$  does:

If  $d + 1 < L(v)$  then  $parent(v) = w$ ;

$L(v) = d + 1$ ;

Send  $Layer(d + 1)$  to all neighbors except  $w$ .

Time complexity:  $O(D)$ .

Message Complexity:  $O(n|E|)$ .

# Distributed DFS

Distributed DFS algorithm:

1. Start exploration (visit) at root  $r$ .
2. When  $v$  is visited for the first time:
  - 2.1 Inform all neighbors of  $v$  that  $v$  has been visited.
  - 2.2 Wait for acknowledgment from all neighbors.
  - 2.3 Resume the DFS process.

The above algorithm ensures that only tree edges are traversed.

Hence time complexity is  $O(n)$ .

Message complexity is  $O(|E|)$ .