Exchanging information on a spanning tree

How to start a rumor with saying as few as possible.

Marco Aiello, Eirini Kaldeli

University of Groningen

Distributed Systems, 2011

1. **Broadcast and convergecast on a tree**
2. **Constructing a spanning tree with a specified root**
   - BFS ST in synchronous and asynchronous models
3. **Constructing a spanning tree with multiple initiators**
4. **The synchronous GHS algorithm for building a MST**

The broadcast and convergecast problem

- Each node has only partial view of the network graph, confined to its neighbours.
- Need to build convenient structures to support efficient information exchange.
- *Broadcast* problem: a processor p_r wants to send a message `<M>` to all other processors.
- *Convergecast* problem: Collect information from all processors to a designated p_r in order to compute a global function (e.g. max value).
- Want algorithms that avoid redundant deliveries and infinite loops.

Message and time complexity

We are concerned about worst case performance:

- *Message complexity*: maximum, over all admissible executions, number of messages sent.
- *Time complexity*: maximum time until termination in any admissible execution
  - Synchronous model (message sent at round t is received at the same round): time complexity amounts to the total number of rounds.
  - Asynchronous model (message sent is eventually received): we assume that the delay of a message is at most one time unit. Since no lower bound on delay is imposed, arbitrary interleaving of events is allowed.

Broadcast on a given spanning tree

- Consider a connected undirected graph topology (bidirectional communication between every pair of neighbours).
- *Spanning Tree*: a subgraph which is a tree and connects all nodes together. The ST of a graph with n nodes has $n-1$ edges.
- Broadcast conducted on a given ST with root p_r:
  - p_r sends <M> to all its children.
  - When a non-leaf p_i receives <M> from its parent, it forwards it to all its children.
- Complexity: $n-1$ messages are sent, in time h, where h is the height of the ST (at most $n-1$, when the ST is a chain).

Convergecast on a given spanning tree

- Initiated by the leaves, usually after a request sent by the root by broadcast
- Each leaf sends its report to its parents.
- Each non-leaf p_i that is not the root waits until it receives reports from all its children, adds its own report, and sends the collective report to its parent.
- When the root receives reports from all its children, it computes the global function on them.
- Time and message complexity is the same as in broadcast.
- So, we'd like a ST with minimum height for efficient broadcast and convergence.

The flooding algorithm

Code for $p_i, 1 \leq i \leq n$

```
parent := ⊥, children := {}

if (p_i = p_r ⋀ parent = ⊥) then        //root sends <M>
 send <M> to all neighbours; parent:=p_i;

Upon receiving <M> from neighbour p_j :
if (parent = ⊥) then        //p_i hasn't received <M> before
 parent:=p_j; send <PARENT> to p_j;
 send <M> to all neighbours except p_j;
else send <ALREADY> to p_j;

Upon receiving <PARENT> from neighbour p_j :
children:=children ∪ {p_j};

if (have received <PARENT> or <ALREADY> from
all neighbours except parent) then terminate;
```

The flooding algorithm: remarks

- In case of receiving <M> for the first time from several processors concurrently, arbitrarily choose one for parent, and send <ALREADY> to others.
- BFS approach: attempt to add all nodes of the same level at a time. In a BFS tree messages are routed along shortest paths.
- Message complexity:
  - Each <M> is sent at most twice on each channel. Actually, <M> is sent twice on all channels, except for those that are part of the ST.
  - So, in total $2l - (n - 1)$ <M> are sent, where l is the number of edges in the network graph, plus $2l - (n - 1)$ respective ack messages: $O(l)$.

- Proceed in atomic rounds: in each round, every p_i sends its pending messages, the messages are delivered and then every p_i processes the messages it has just received.
- Each node at distance t from the root receives `<M>` at round t.
- The outcome is a BFS ST, i.e. one whose height h equals the graph's diameter d.
- Time complexity for ST construction: $O(d)$.
- *Broadcast* from the root (and convergecast to the root) on the resulting ST is time optimal: $O(d)$.

The flooding algorithm: asynchronous case

- By time t <M> has reached all p_is that are at distance t or less from p_r.
- Thus, time complexity for the ST construction is again $O(d)$.
- How about the time complexity of broadcast on the resulting ST, i.e. what's its height?
- <M> may propagate more quickly on some channels than others.
- In an asynchronous model the flooding algorithm doesn't necesserily construct a BFS tree.
- In worst case, the ST is a chain with height $n - 1$.

Ensuring shortest path ST for the asynchronous case

- Can also transmit information about the number of hops *dist* of <M> so far.
- Each p_i has a variable *my_dist* that keeps the shortest path along which it has received <M>. Initially $my\_dist = \infty$.
- Upon receiving <M, *dist*> from p_j:
  - if $my\_dist > dist + 1$ then $parent := p_j$, $my\_dist = dist + 1$, send <PARENT> to p_j and <M, *my_dist*)> to the rest of the neighbours.
  - if $my\_dist = dist + 1$ then send <ALREADY> to p_j.
  - if $my\_dist < dist + 1$ do nothing (already sent <M, *my_dist*> to p_j).
- Complexity: $O(d)$ time and $O(nl)$ messages (each p_i reduces its *my_dist* at most $n - 1$ times).

Spanning tree without assuming a single root

- Any p_i may spontaneously intiate the flooding algorithm to construct a ST with itself as a root.
- If a node receives requests to join some STs with different initiators, the node has to commit to only one instance by applying some tie-breaking rule to make a choice.
- Assuming that each p_i has a unique natural number identifier *id*, commit to the ST whose root has the highest *id*.
- Adapt the flooding algorithm to accommodate for concurrent initiators:
  - Messages are piggybacked with the root *id*.
  - Each p_i has a variable *my_root* that keeps the maximal *id* seen so far. Initially $my\_root = -1$.

Multiple initiators flooding algorithm

- When p_i receives <M(*id*)> from p_j:
  - If $my\_root < id$ then p_i discards its current execution, re-initialises its data structure ($my\_root := id$ and $parent := p_j$) and joins *id*'s ST execution.
  - If $my\_root = id$ then p_j's execution is initiated by the same root as p_i's and p_i has already identified its parent, so just send <ALREADY(id)> to p_j.
  - If $my\_root > id$ then again send <ALREADY(*id*)> to p_j, since p_j will eventually receive <M(*my_root*)> and connect to *my_root*'s tree as well.
- <ALREADY(*id*)> and <PARENT(*id*)> are ignored unless $id = my\_root$.
- Complexity: $O(ln)$ messages, $O(d)$ time.

- Assume each edge (i, j) is associated with a non-negative real-valued *weight*, known to both i and j.
- Weights can represent latency, traffic load, bandwidth etc. The *weight* of a subgraph is the sum of the weights of its edges.
- Want to build a minimum-weight spanning tree (MST): minimize the cost for any source p_i to communicate with all other processes.
- *Synchronous GHS* (Gallagher, Humblet and Spira) algorithm: start with the trivial spanning forest of n nodes and no edges and repeatedly connect components along minimum-weight outgoing edges (MWOE).
- Assume that each edge has a distinct weight: this guarantees that the MST is unique.

Addition of MWOEs yields a MST

Lemma

For any spanning forest $\{(N_i, L_i) | i = 1 \ldots k\}$ of a weighted undirected graph G, consider any component (N_j, L_j). Denote by λ_j the edge with the smallest weight among those that have exactly one endpoint in N_j. Then an MST for G that includes all edges in each L_j in the spanning forest must also include λ_j.

- Proof sketch: If for any component (N_j, L_j) instead of λ_j another outgoing edge λ' is used, then the resulting tree cannot be a MST, because the replacement of λ' by λ yields a lower cost tree.
- Thus, if we start with a forest all of whose edges are in the unique MST, then the MWOEs of all components are also in the MST.

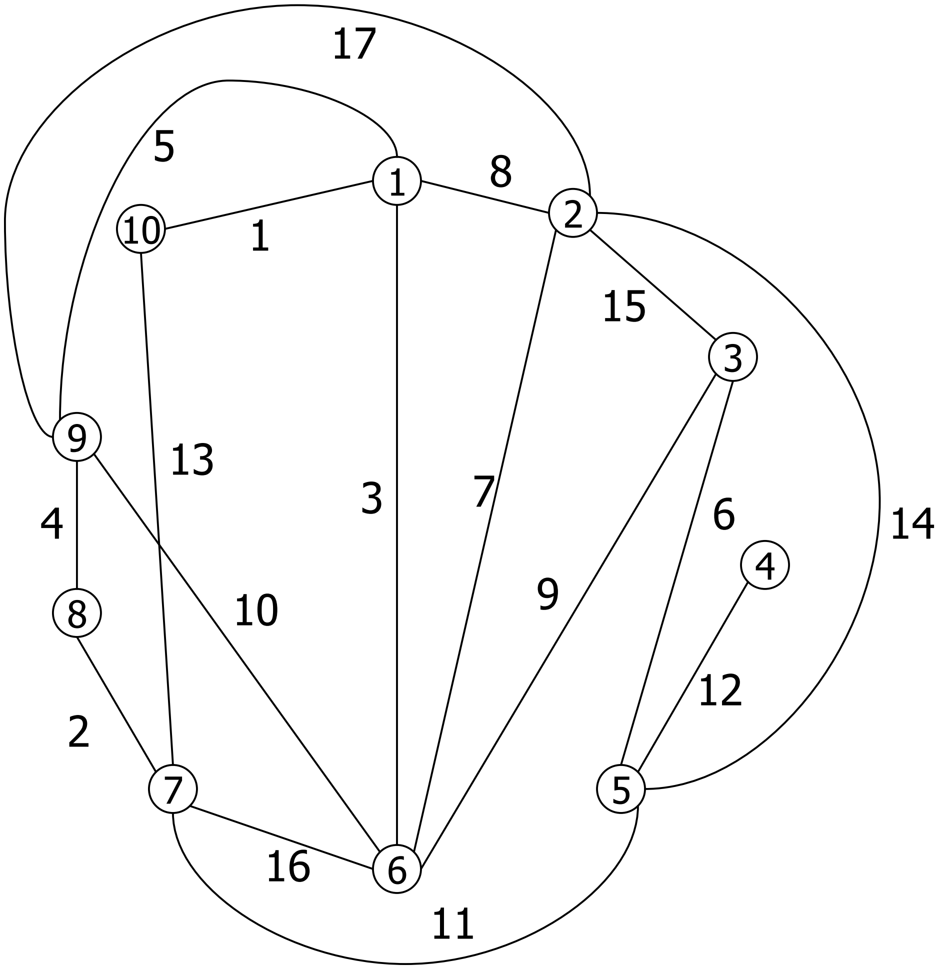
Synchronous GHS algorithm

- The addition of MWOEs is done concurrently by all components.
- Proceed in levels: the level k components constitute a spanning forest, where each component consists of a tree that is a subgraph of the MST.
- Each component, at every level, has a distinguished node, the *leader*. All nodes know their component's leader's *id lID*.
- Each p_i knows which of its incident edges belong to the component's MST.

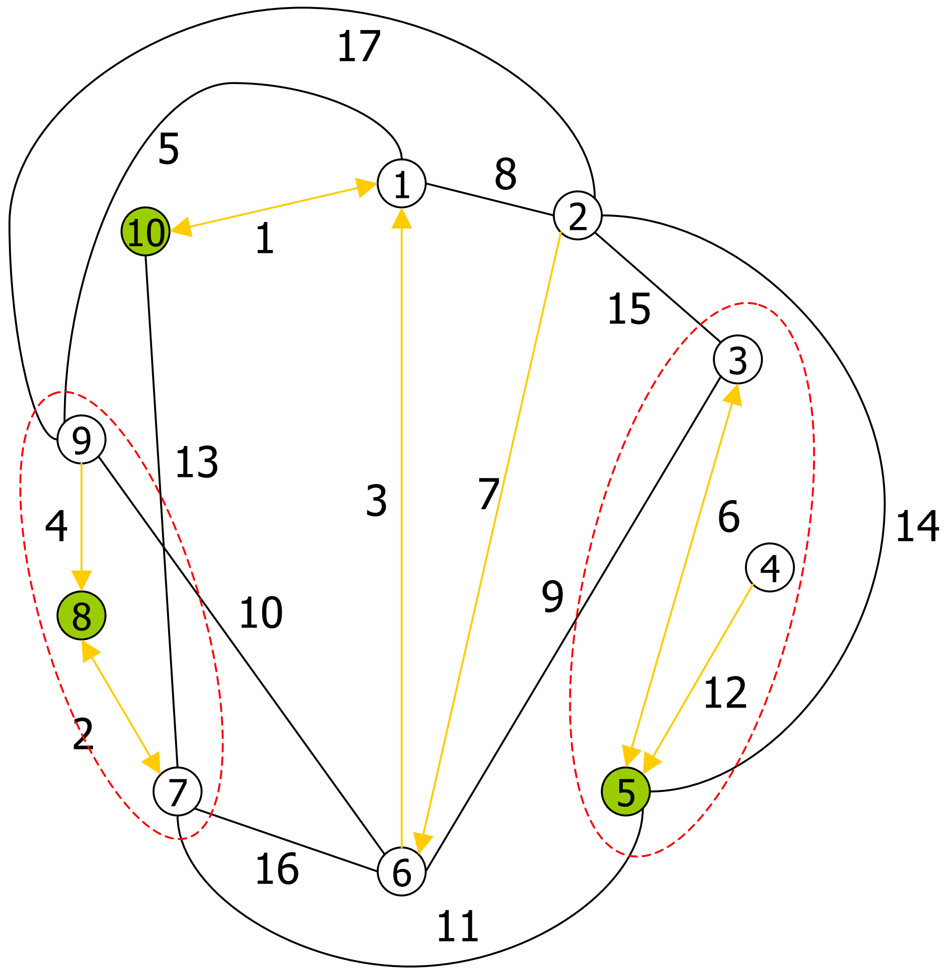
Synchronous GHS algorithm: determine the MWOE of each component

- Initially, the level 0 components consist of n single nodes and no edges.
- Suppose inductively that the level k components have been determined.
- To construct the $k+1$ level components each level k component identifies its MWOE through a broadcast-convergecast-broadcast sequence (performed as described in the previously presented algorithms).
- The component leader broadcasts search request *SEARCH_MWOE*(*lID*) along the *tree edges*.
- Upon receiving *SEARCH_MWOE*(*lID*), each p_i sends test messages along all non-tree edges to find out if the other end belongs to the same component (has the same leader).

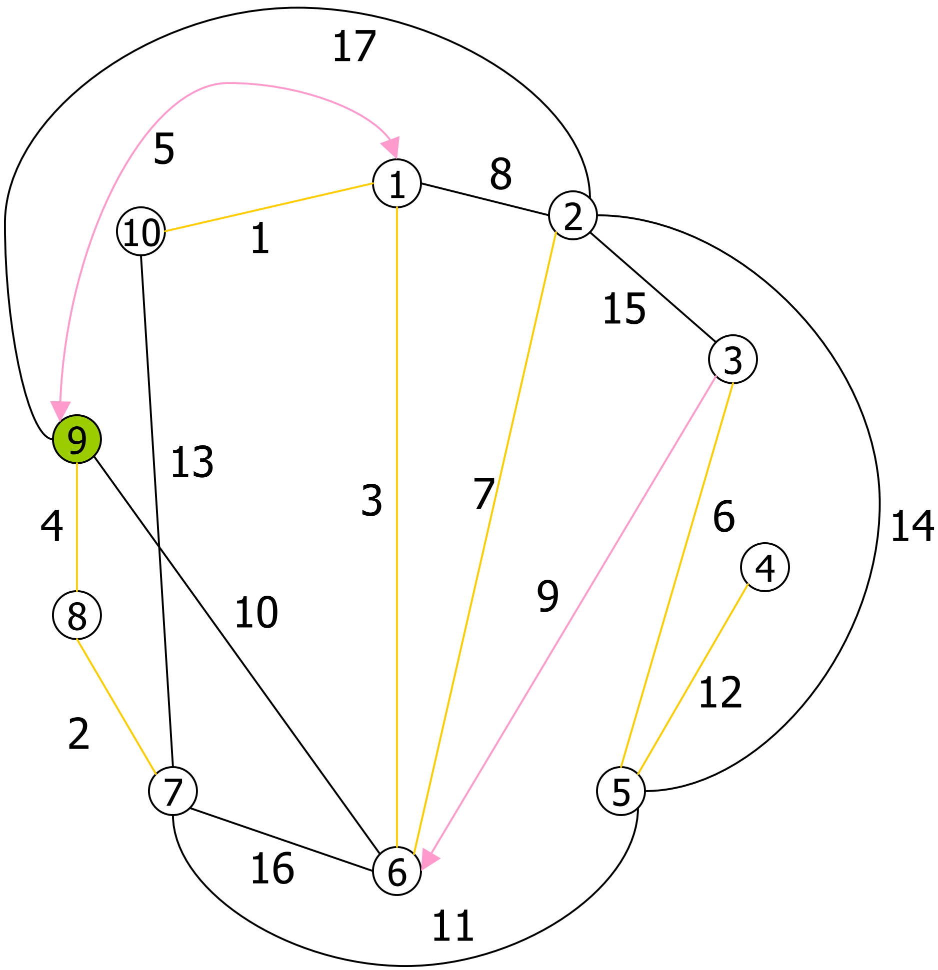
Minimum Spanning Tree (continued)



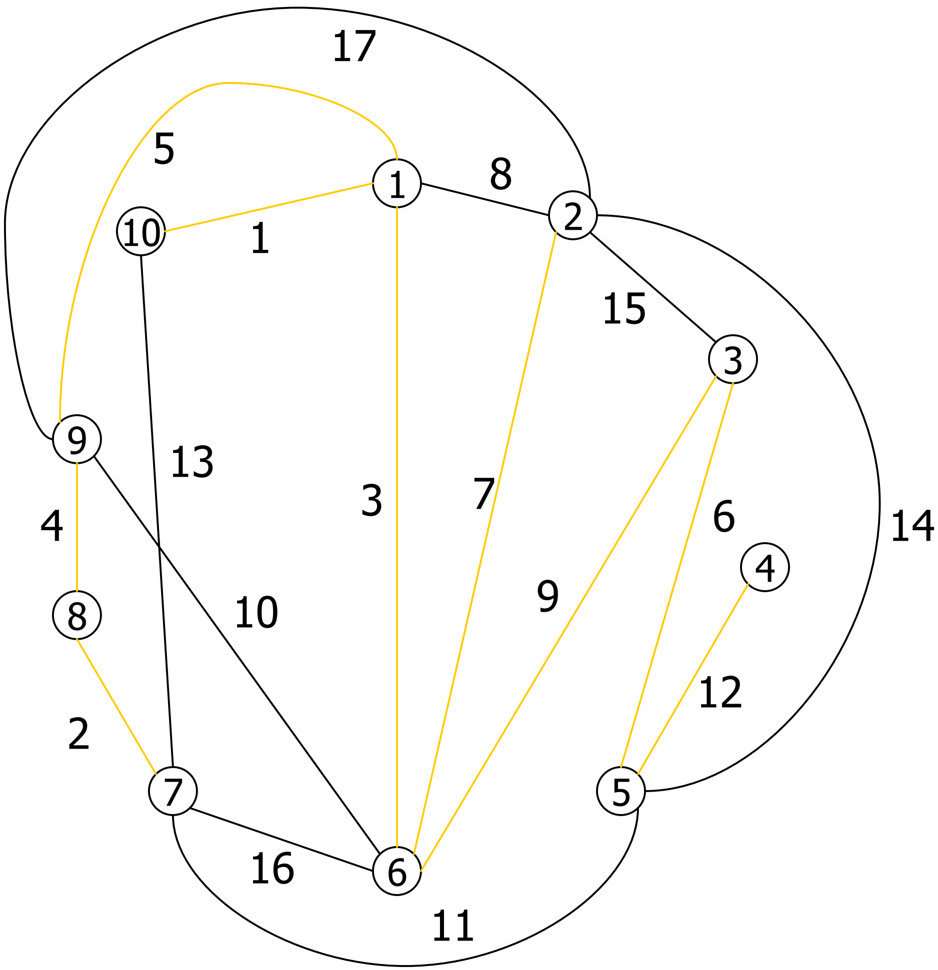
Minimum Spanning Tree (continued)



Minimum Spanning Tree (continued)



Minimum Spanning Tree (continued)



Synchronous GHS algorithm: determine the MWOE of each component

- Each p_i selects among all its incident edges that are outgoing from the component the one with the minimum weight $local\_MWOE(i, j)$.
- The $local\_MWOE(i, j)$ information is convergecasted to the leader, taking minima along the way.
- The leader obtains the overall minimum $MWOE(localID, remoteID)$ and broadcasts $ADD\_MWOE(localID, remoteID)$ along the tree edges. The node whose *id* is *localID* marks edge $(localID, remoteID)$ as belonging to the component's MST.

Synchronous GHS algorithm: Merging k level components

A new leader has to be chosen for each group of level k components that have been combined to a single level $k + 1$ component.

Lemma

In each new component at level level $k + 1$, there is exactly one edge that is the MWOE chosen by two of the merged level k components.

- Proof: based on the fact that weights are distinct.
- This allows the computation of a unique new leader by just finding the common MWOE of two merged components.

Synchronous GHS algorithm: Merging k level components

- If a MWOE is marked by both components on which it is incident, then the endpoint of the common MWOE with the larger *id* becomes the new leader: $\mathit{lID} = \mathit{max}(\mathit{localID}, \mathit{remoteID})$.
- The new leader broadcasts a *NEW_LEADER*(*lID*) message along the MST of the newly formed component, so that all nodes in the $k+1$ round know their new leader.
- The algorithm terminates when leader learns that no MWOE can be found by any node in its component.

Synchronous GHS algorithm: time complexity

- Levels have to be kept synchronized: when p_i queries p_j whether it belongs to the same component, both p_i and p_j must have up-to-date *lid*s (guarantee that last *NEW_LEADER*(*lID*) has reached them).
- To ensure this, each level comprises a fixed number of rounds :$O(n)$ (for this purpose, each p_i should be aware of n).
- At each level k, each component is combined with at least one other component, starting from the single node components.
- Thus, a level k component comprises at least 2^k nodes, so the number of levels is at most $\log n$.
- Time complexity: $O(n \log n)$

Synchronous GHS algorithm: message complexity

- At each level $O(n)$ messages are sent for the broadcasts and convergecast along the tree edges, plus $O(l)$ to identify the local MWOE: $O((n + l) \log n)$
- Can do a little better for the messages required for the local MWOE determination:
  - Each node marks its incident edges that lead to a node of the same component as "rejected"; thereafter, there is no need to test them again: each edge gets tested and rejected at most once, so $O(l)$ messages in total.
  - If each node tests the remaining candidate edges one at a time in order of increasing weight, then only one message per level is required for each node: $O(n \log n)$ messages in total.
- Improved message complexity (optimal): $O(n \log n + l)$

Synchronous GHS algorithm: non-unique edge weights

- What if edge weights are not distinct?
- This would cause a problem because we have to find a way to break ties, while the lemma about the unique edge that is the common MWOE of two of the merging components doesn't hold.
- Introduce unique *edge identifiers*: *id* of edge (i, j) is the triple $(weight_{(i,j)}, id_i, id_j)$.
- Compare the edge identifiers instead of the weights, based on the lexicographic total order among the triples.