Sync 1-initiator ST (flooding)

```
(local variables)
int visited, depth ⟵ 0
int parent ⟵ ⊥
set of int Neighbors ⟵ set of neighbors
(message types)
QUERY

(1) if i = root then
(2)       visited ⟵ 1;
(3)       depth ⟵ 0;
(4)       send QUERY to Neighbors;
(5) for round = 1 to diameter do
(6)       if visited = 0 then
(7)             if any QUERY messages arrive then
(8)                   parent ⟵ randomly select a node from which QUERY was received;
(9)                   visited ⟵ 1;
(10)                  depth ⟵ round;
(11)                  send QUERY to Neighbors \ {senders of QUERYs received in this round};
(12)      delete any QUERY messages that arrived in this round.
```

Synchronous 1-init Spanning Tree: Example

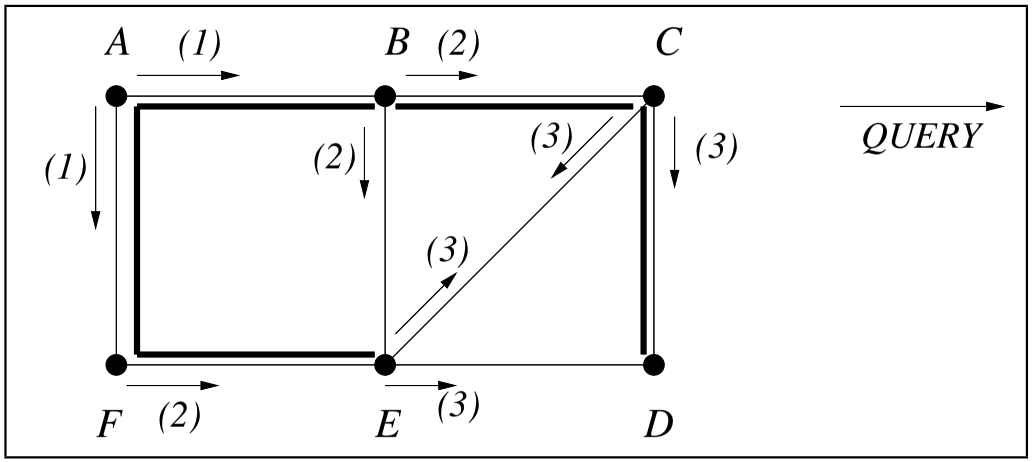

Figure 5.2: Tree in boldface; round numbers of QUERY are labeled

- Designated root. Node A in example.
- Each node identifies parent
- How to identify child nodes?

Synchronous 1-init Spanning Tree: Complexity

Termination: after *diameter* rounds.
How can a process terminate after setting its *parent*?

Complexity:

- Local space: $O(degree)$
- Global space: $O(\sum \text{local space})$
- Local time: $O(degree + diameter)$
- Message time complexity: d rounds or message hops
- Message complexity: $\geq 1, \leq 2$ messages/edge. Thus, $[l, 2l]$

Spanning tree: analogous to breadth-first search

Asynchronous 1-init Spanning Tree: Code

(local variables)
int *parent* $\longleftarrow \perp$
set of int *Children*, *Unrelated* $\longleftarrow \emptyset$
set of int *Neighbors* $\longleftarrow$ set of neighbors
(message types)
QUERY, ACCEPT, REJECT

(1) When the predesignated root node wants to initiate the algorithm:
(1a) **if** ($i = root$ **and** $parent = \perp$) **then**
(1b) **send** QUERY to all neighbors;
(1c) $parent \longleftarrow i$.

(2) When QUERY arrives from j:
(2a) **if** $parent = \perp$ **then**
(2b) $parent \longleftarrow j$;
(2c) **send** ACCEPT to j;
(2d) **send** QUERY to all neighbors except j;
(2e) **if** $(Children \cup Unrelated) = (Neighbors \setminus \{parent\})$ **then**
(2f) **terminate**.
(2g) **else send** REJECT to j.

(3) When ACCEPT arrives from j:
(3a) $Children \longleftarrow Children \cup \{j\}$;
(3b) **if** $(Children \cup Unrelated) = (Neighbors \setminus \{parent\})$ **then**
(3c) **terminate**.

(4) When REJECT arrives from j:
(4a) $Unrelated \longleftarrow Unrelated \cup \{j\}$;
(4b) **if** $(Children \cup Unrelated) = (Neighbors \setminus \{parent\})$ **then**
(4c) **terminate**.

Async 1-init Spanning Tree: Operation

- root initiates flooding of QUERY to identify tree edges
- *parent*: 1st node from which QUERY received
  - ACCEPT (+ rsp) sent in response; QUERY sent to other nbhs
  - Termination: when ACCEPT or REJECT (- rsp) received from non-parent nbhs. Why?
- QUERY from non-parent replied to by REJECT
- Necessary to track neighbors? to determine children and when to terminate?
- Why is REJECT message type required?
- Can use of REJECT messages be eliminated? How? What impact?

Asynchronous 1-init Spanning Tree: Complexity

Local termination: after receiving ACCEPT or REJECT from non-parent nbhs.
Complexity:

- Local space: $O(degree)$
- Global space: $O(\sum \text{local space})$
- Local time: $O(degree)$
- Message complexity: $\geq 2, \leq 4$ messages/edge. Thus, $[2l, 4l]$
- Message time complexity: $d + 1$ message hops.

Spanning tree: no claim can be made. Worst case height $n - 1$