Spanning Tree Construction

A spanning tree $T$ of a graph $G = (V,E)$ is an acyclic subgraph of $G$ such that $T=(V,E')$ and $E' \subseteq E$.

Assumptions:
- single initiator
- bidirectional links
- total reliability
- $G$ connected

Protocol SHOUT

Initially: $\forall x$, Tree-neighbors($x$) = \{

At the end:
$\forall x$, Tree-neighbors($x$) = \{\text{links that belong to the spanning tree}\}

Example

1. \text{init}
   \begin{align*}
   Q? &= \text{do you want to be my neighbour in the spanning tree?} \\
   Q? &= \text{yes, if it is the first time; } \\
   Q? &= \text{no, if I have already answered yes to someone else:}
   \end{align*}

2. $Q$
States $S:=$ (INITIATOR, IDLE, ACTIVE, DONE)
$S_{init}:=$ (INITIATOR, IDLE)
$S_{term}:=$ (DONE)

INITIATOR
Spontaneously
- $root:=\text{true}$
- $\text{Tree-neighbours} := \{\} $
- $send(Q) \text{ to } N(x)$
- $counter:=0$
- $\text{become ACTIVE}$

IDLE
$receiving(Q)$
- $root:=\text{false}$
- $parent:=\text{sender}$
- $\text{Tree-neighbours} := \{\text{sender}\}$
- $send(\text{yes}) \text{ to } \text{sender}$
- $counter:=1$
- if $counter = |N(x)|$ then $\text{become DONE}$
- else $send(Q) \text{ to } N(x) - \{\text{sender}\}$
- $\text{become ACTIVE}$

ACTIVE
$receiving(Q)$
- $send(\text{no}) \text{ to } \text{sender}$

$receiving(\text{yes})$
- $\text{Tree-neighbours} := \text{Tree-neighbours} \cup \text{sender}$
- $counter:=counter+1$
- if $counter = |N(x)|$ then $\text{become DONE}$

$receiving(\text{no})$
- $counter:=counter+1$
- if $counter = |N(x)|$ then $\text{become DONE}$

Correctness and Termination
- If $x$ is in $\text{Tree-neighbours}$ of $y$, $y$ is in $\text{Tree-neighbours}$ of $x$
- If $x$ send YES to $y$, then $x$ is in $\text{Tree-neighbour}$ of $y$ and is connected to the initiator by a chain of YES
- Every $x$ (except the initiator) sends exactly one YES

The spanning graph defined by the $\text{Tree-neighbour}$ relation is connected and contains all the entities

Note: local termination
**Message Complexity**

\[ \text{SHOUT} = \text{FLOOD} + \text{REPLY} \]

\[ \text{Messages(SHOUT)} = 2 \cdot M(\text{FLOOD}) \]

**Possible situations**

- \( Q \rightarrow \text{yes} \)
- \( Q \rightarrow \text{no} \)
- \( \text{no} \rightarrow \text{no} \)

**Impossible situations**

- \( \text{no} \rightarrow \text{yes} \)
- \( \text{yes} \rightarrow \text{yes} \)

**Message Complexity - worst case**

**Total n. of Q:**

- \( Q \rightarrow \text{yes} \)
  - \( (n-1) \)
- \( Q \rightarrow \text{no} \)
  - \( m - (n-1) \)

  - only one Q on the ST links
  - on the other links

**Total:**

\[ 2(m - (n-1)) + (n-1) \]

\[ = 2m - n + 1 \]

**Total n. of NO:**

- \( \text{no} \rightarrow \text{no} \)

**Total n. of YES:**

- \( \text{yes} \rightarrow \text{yes} \)

**as many as Q \rightarrow Q**

\[ 2(m - (n-1)) \]

**Exactly:** \( (n-1) \)
Message Complexity - worst case

\[ 2m - n + 1 + 2(m - (n-1)) + n-1 \]
\[ = 2m -n +1+2m -2n +2 + n - 1 \]
\[ = 4m -2n + 2 \]

\[ \text{Messages(SHOUT)} = 4m -2n + 2 \]

In fact: \( M(SHOUT) = 2 M(FLOOD) = 2(2m-n+1) \)

\( \Omega(m) \) is a lower bound also in this case

Spanning Tree Construction

Without "NO"

Protocol SHOUT+

States \( S = \{\text{INITIATOR, IDLE, ACTIVE, DONE}\} \)
\( S_{init} = \{\text{INITIATOR, IDLE}\} \)
\( S_{term} = \{\text{DONE}\} \)

INITIATOR

Spontaneously

\begin{align*}
\text{root} &:= \text{true} \\
\text{Tree-neighbours} &:= () \\
\text{send}(Q) &\text{ to } N(x) \\
\text{counter} &:= 0 \\
\text{become ACTIVE}
\end{align*}

IDLE

\begin{align*}
\text{receiving}(Q) &\quad \text{root} := \text{false} \\
\text{parent} &:= \text{sender} \\
\text{Tree-neighbours} &:= (\text{sender}) \\
\text{send}(\text{yes}) &\text{ to } \text{sender} \\
\text{counter} &:= 1 \\
\text{if} \ \text{counter} = |N(x)| \text{ then} &\quad \text{become DONE} \\
\text{else} &\quad \text{send}(Q) \text{ to } N(x) - (\text{sender}) \\
\text{become ACTIVE}
\end{align*}

ACTIVE

\begin{align*}
\text{receiving}(Q) \ (\text{to be interpreted as NO}) &\quad \text{counter} := \text{counter} + 1 \\
\text{if} \ \text{counter} = |N(x)| &\quad \text{become DONE} \\

\text{receiving}(\text{yes}) &\quad \text{Tree-neighbours} := \text{Tree-neighbours} \cup (\text{sender}) \\
\text{counter} &:= \text{counter} + 1 \\
\text{if} \ \text{counter} = |N(x)| &\quad \text{become DONE}
\end{align*}
On each link there will be exactly 2 messages:

\[ \begin{align*}
    &Q \rightarrow \text{yes} \\
    \text{or} &Q \rightarrow \text{yes}
\end{align*} \]

Either

\[ Messages(SHOUT+) = 2m \]

or

\[ Messages(SHOUT) = 4m - 2n + 2 \]

Much better than:

Spanning Tree Construction

With Notification

States \( S = \{ \text{INITIATOR, IDLE, ACTIVE, DONE} \} \)
\( S_{init} = \{ \text{INITIATOR, IDLE} \} \)
\( S_{term} = \{ \text{DONE} \} \)

**INITIATOR**

*Spontaneously*

- root := true
- Tree-neighbours := \{ \}
- send(Q) to N(x)
- counter := 0
- ack-counter := 0
- become ACTIVE

**IDLE**

*receiving(Q)*

- root := false
- parent := sender
- Tree-neighbours := \{ sender \}
- send(yes) to sender
- counter := 1
- ack-counter := 0
- if counter = |N(x)| then
  - CHECK
- else
  - send(Q) to N(x) - \{ sender \}
  - become ACTIVE

**ACTIVE**

*receiving(Q)*

- counter := counter + 1
- if counter = |N(x)| and not root then
  - CHECK

*receiving(yes)*

- Tree-neighbours :=
  - Tree-neighbours \cup \{ sender \}
- counter := counter + 1
- if counter = |N(x)| and not root then
  - CHECK
ACTIVE (cont)

\textbf{receiving(Ack)}

\begin{align*}
\text{ack-counter} & := \text{ack-counter} + 1 \\
\text{if} \ |\text{counter}| = |\text{N}(x)| & /* \text{indicate tree-neighbors is done} \\
\text{if} \ \text{root} & \\
\text{if} \ \text{ack-counter} = |\text{Tree-neighbors}| & \\
\text{send}(\text{Terminate}) & \text{to} \ \text{Tree-neighbors} \\
& \text{become DONE} \\
\text{else if} \ \text{ack-counter} = |\text{Tree-neighbors}| - 1 & \\
\text{send}(\text{Ack}) & \text{to} \ \text{parent} \\
\end{align*}

\textbf{receiving(Terminate)}

\begin{align*}
\text{send}(\text{Terminate}) & \text{to} \ \text{Children} \\
& \text{become DONE} \\
\end{align*}

\textbf{CHECK}

\text{If} \ \text{I am a leaf} \\

\begin{align*}
\text{(* that is: Children:= Tree-neighbours - (parent)} & \\
\text{if Children = emptyset *)} \\
\text{send(Ack) to parent} \\
\end{align*}

What happens if there are multiple initiators?

An election is needed to have a unique initiator.

or

Another protocol has to be devised.

\textbf{NOTE: Election is impossible if the nodes do not have distinct IDs}
**Traversal**
*Depth First Search*

Assumptions
- Single initiator
- Bidirectional links
- No faults
- G connected

\[ S = \{\text{INITIATOR, SLEEPING, ACTIVE, DONE}\} \]

---

**One version**

1) When first visited, remember who sent, forward the token to one of the unvisited neighbours wait for its reply
2) When neighbour receives,
   - if already visited, it will return the token saying it is a back edge
   - otherwise, will forward it (sequentially) to all its unvisited neighbour before returning it
3) If there are no more unvisited neighbours, return the token (reply) to the node from which it first received the token
4) Upon reception of reply, forward the token to another unvisited neighbour

**Complexity**

*Message Complexity:*

Type of messages: token, back, return

\[ 2m = O(m) \]

\[ \Omega(m) \text{ is also a lower bound} \]

---

*Time Complexity:*

(ideal time) \[ 2m = O(m) \]  
Totally sequential
Note:  
most messages are on Back Edges

--- most time is spent on Back Edges

Idea: avoid sending messages on back edges

How?

DF+ Complexity

<table>
<thead>
<tr>
<th>Message</th>
<th>Time (ideal time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token, Return, Visited, Ack (ok)</td>
<td>2(n-1)</td>
</tr>
<tr>
<td>Each entity (except init): receives 1 Token, sends 1 Return:</td>
<td>2(n-1)</td>
</tr>
</tbody>
</table>
| Each entity:  
  1 Visited to all neighbours except the sender | 2n               |
| \[|N(s)| \sum (|N(x)|-1) \] | TOT: 4n -2        |
| + (same for Ack) | TOT: 4m           |

DF+ Improving Time
**Summarizing:**

<table>
<thead>
<tr>
<th></th>
<th>Messages</th>
<th>Ideal Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF:</td>
<td>2m</td>
<td>2m</td>
</tr>
<tr>
<td>DF+:</td>
<td>4m</td>
<td>4n -2</td>
</tr>
</tbody>
</table>

**DF Traversal**

A token is sent to an already visited node (= back edge)
Both nodes will eventually understand the "mistake"
and pretend nothing happened

**DF++ Complexity**

In the worst case there is a "mistake" on each link except for the tree links

\[
\text{Messages} = 4m -(n-1)
\]

But when we measure ideal time:

"mistakes" will not happen

\[
\text{Time} = 2(n-1)
\]

**DF+++**

Do not send the Ack
What happens?

**Summary**

<table>
<thead>
<tr>
<th></th>
<th>Messages</th>
<th>Ideal Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF:</td>
<td>2m</td>
<td>2m</td>
</tr>
<tr>
<td>DF+:</td>
<td>4m</td>
<td>4n -2</td>
</tr>
<tr>
<td>DF++</td>
<td>4m-n+1</td>
<td>2n+1</td>
</tr>
</tbody>
</table>
Observations

Time ...

Termination ...

An application:
  access permission problems, e.g., Mutual Exclusion

Any Traversal does a Broadcast (not very efficient)
The reverse is not true.

Another Traversal: Smart Traversal

1- Build a Spanning Tree with SHOUT+
   Time ≤ d+1          d: diameter

2- Perform DF Traversal
   Time = 2(n-1)

   Total Time ≤ 2n+d-1

Summary

<table>
<thead>
<tr>
<th></th>
<th>Messages</th>
<th>Ideal Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF:</td>
<td>2m</td>
<td>2m</td>
</tr>
<tr>
<td>DF+:</td>
<td>4m</td>
<td>4n -2</td>
</tr>
<tr>
<td>DF++</td>
<td>4m-n+1</td>
<td>2n+1</td>
</tr>
<tr>
<td>Smart</td>
<td>2m+2n-2</td>
<td>2n+d-1</td>
</tr>
</tbody>
</table>
Computations with Multiple initiator: WAKE-UP

General FLOOD algorithm: \( O(m) \)

More precisely: \( 2m - n + k^* \)

WHY?

1 init = broadcast = \( 2m - n + 1 \)

All init = \( 2m \)

FLOOD solves the problem.

In special topologies?

FLOOD is optimal

COMPLETE GRAPH

\( \Omega(n^2) \) \quad \Omega(n)

HYPERCUBE

\( O(n \log n) \) \quad \Omega(n \log n)

Broadcast Specific

Wakeup Specific

Need additional assumptions to reduce the complexity