Introduction to the CONGEST Model

Algorithms for Distributed Systems

Sebastian Forster

University of Salzburg



This work is licensed under a Creative Commons Attribution 4.0 International License.

Congestion



Model

- The network is modeled as an undirected, connected graph $G = (V, E)$
  - Nodes V model processors, and edges E model (symmetric) communication links
  - $n = |V|$, $m = |E|$
- Round-based communication:
  - Synchronization by a global clock
  - In each round, every node may send one message to each neighbor
  - Round structure: receive (omitted in round 1) – compute – send
- Nodes have unique IDs, known to themselves and their neighbors
- Non-uniform: n is global knowledge
- Termination occurs when all nodes have reached a termination state

Model

- The network is modeled as an undirected, connected graph $G = (V, E)$
  - Nodes V model processors, and edges E model (symmetric) communication links
  - $n = |V|$, $m = |E|$
- Round-based communication:
  - Synchronization by a global clock
  - In each round, every node may send one message to each neighbor
  - Round structure: receive (omitted in round 1) – compute – send
- Nodes have unique IDs, known to themselves and their neighbors
- Non-uniform: n is global knowledge
- Termination occurs when all nodes have reached a termination state

CONGEST Model:

- Message size: $O(\log n)$ bits
  Bandwidth limitation
- ID length: $O(\log n)$ bits

Model

- The network is modeled as an undirected, connected graph $G = (V, E)$
  - Nodes V model processors, and edges E model (symmetric) communication links
  - $n = |V|$, $m = |E|$
- Round-based communication:
  - Synchronization by a global clock
  - In each round, every node may send one message to each neighbor
  - Round structure: receive (omitted in round 1) – compute – send
- Nodes have unique IDs, known to themselves and their neighbors
- Non-uniform: n is global knowledge
- Termination occurs when all nodes have reached a termination state

CONGEST Model:

- Message size: $O(\log n)$ bits
  Bandwidth limitation
- ID length: $O(\log n)$ bits

LOCAL Model:

- Message size: unlimited
  No bandwidth limitation
- ID length: unlimited

Plan for Today

Algorithmic techniques for the CONGEST model:

1. Broadcast
2. Breadth-first search tree
3. Aggregation
4. Pipelining
5. Queuing

Broadcast



Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

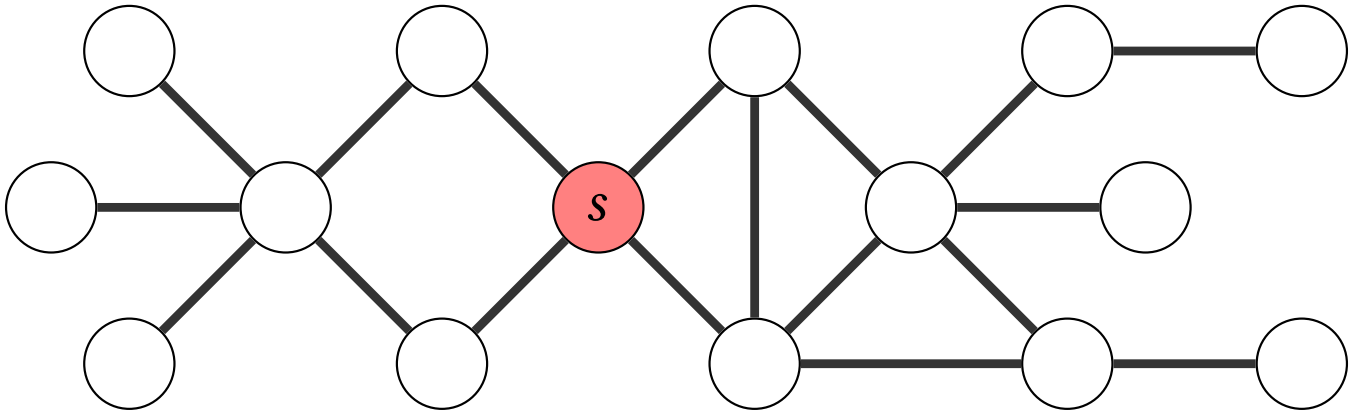
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

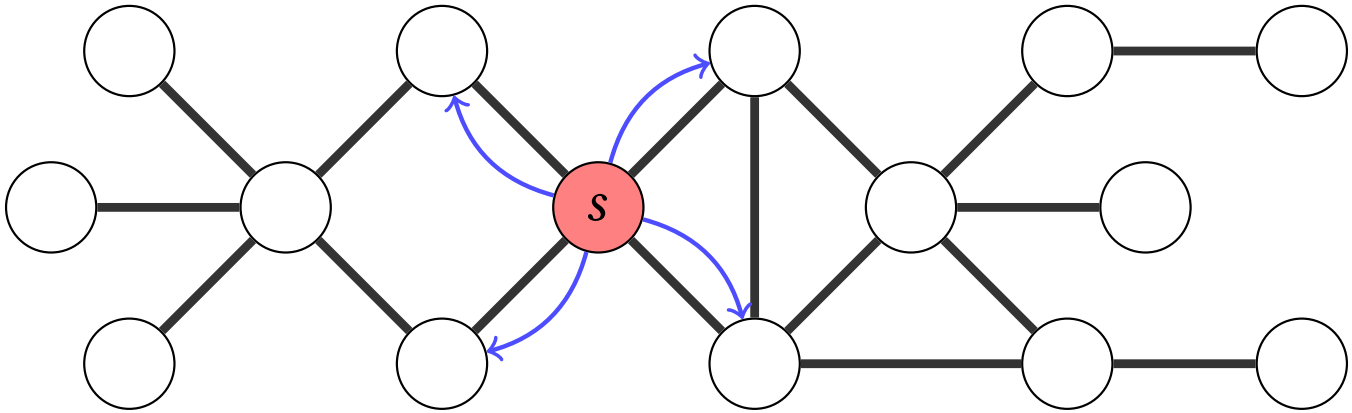


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

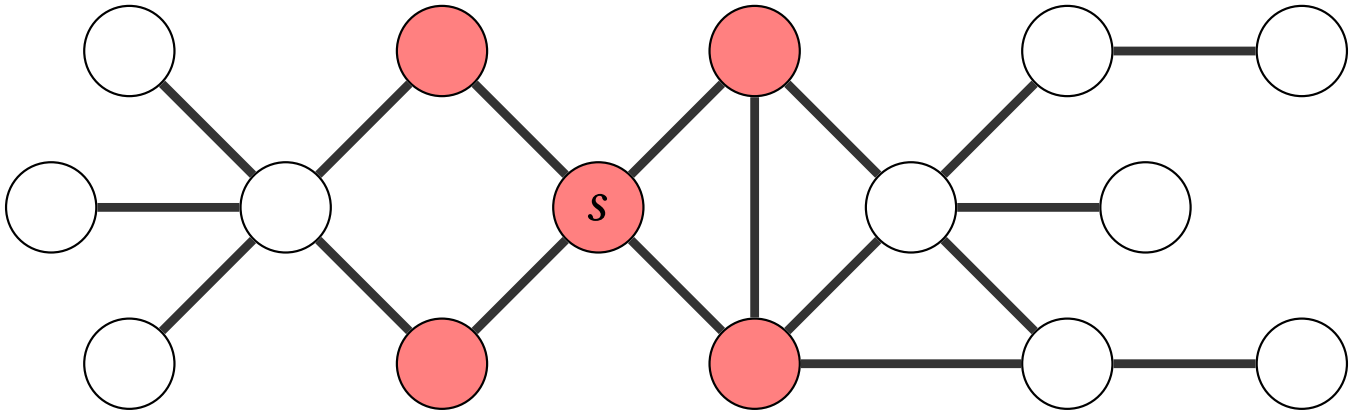


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

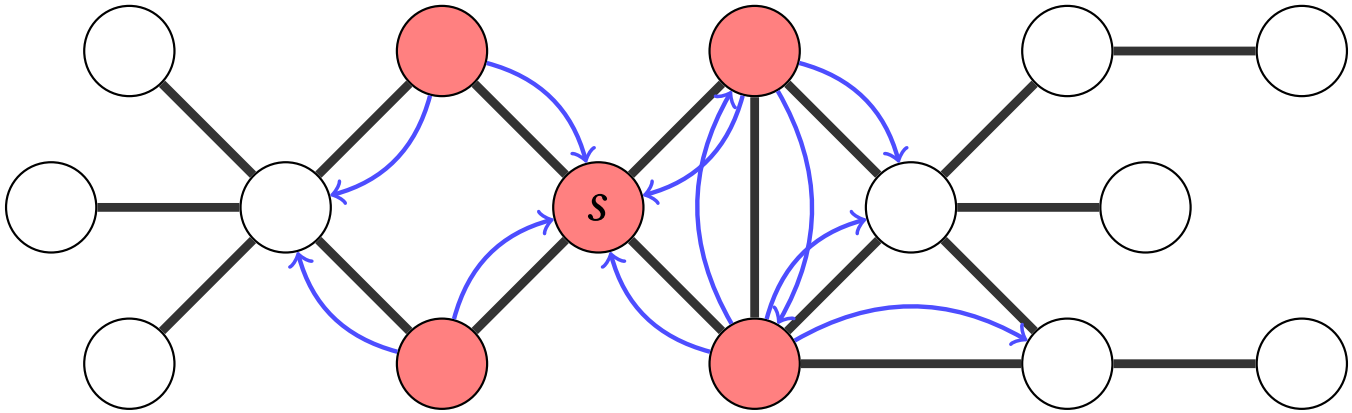


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

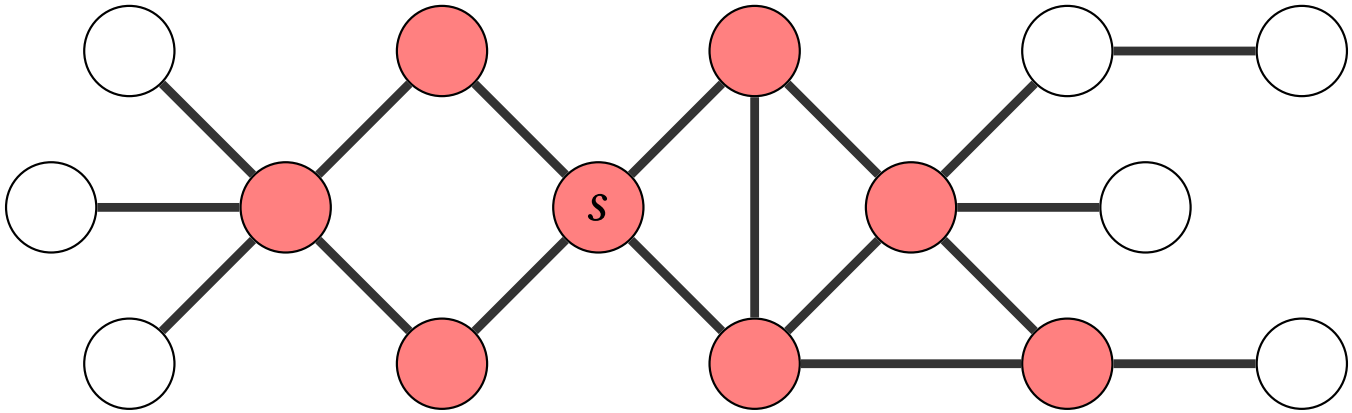


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

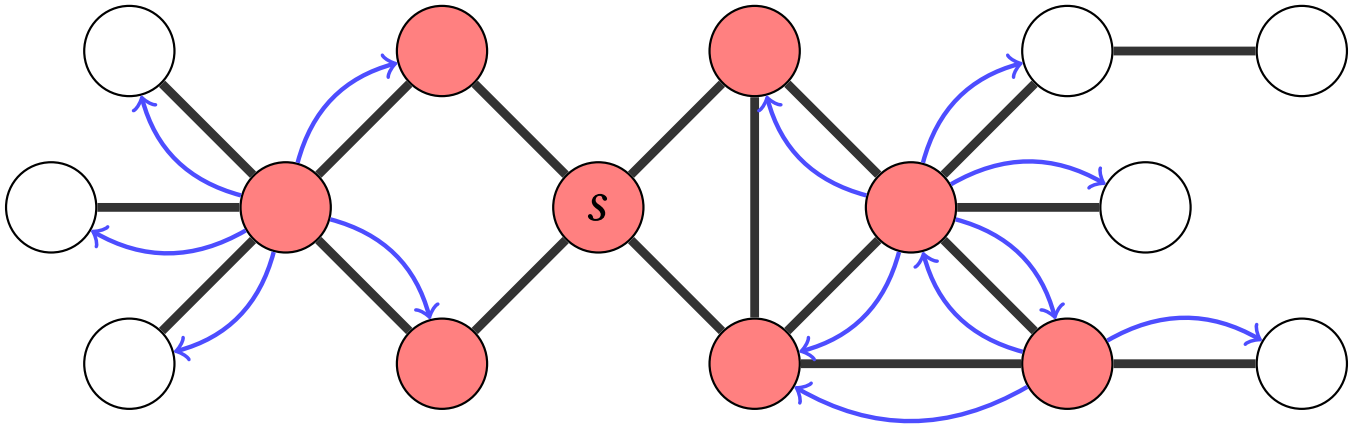


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

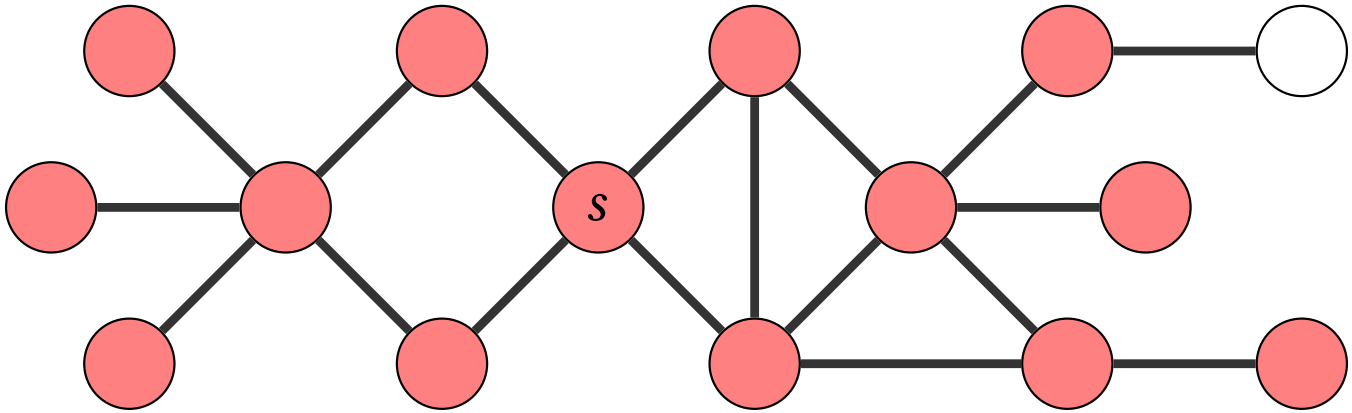


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

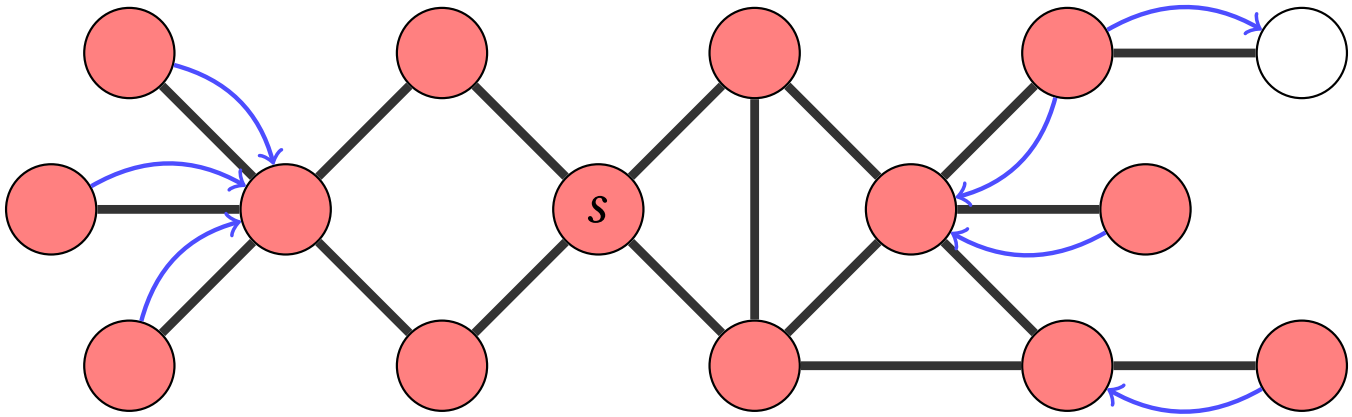


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

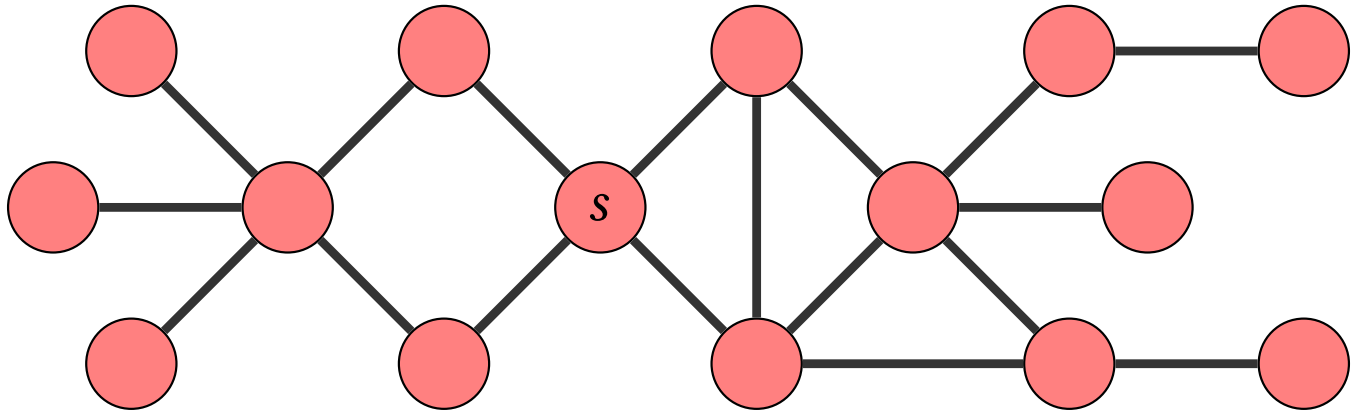


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

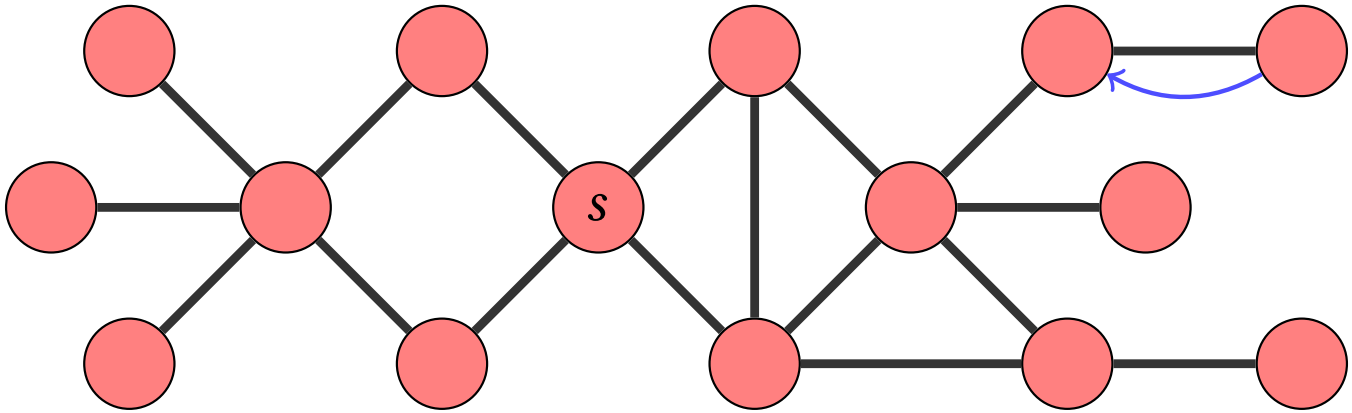


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

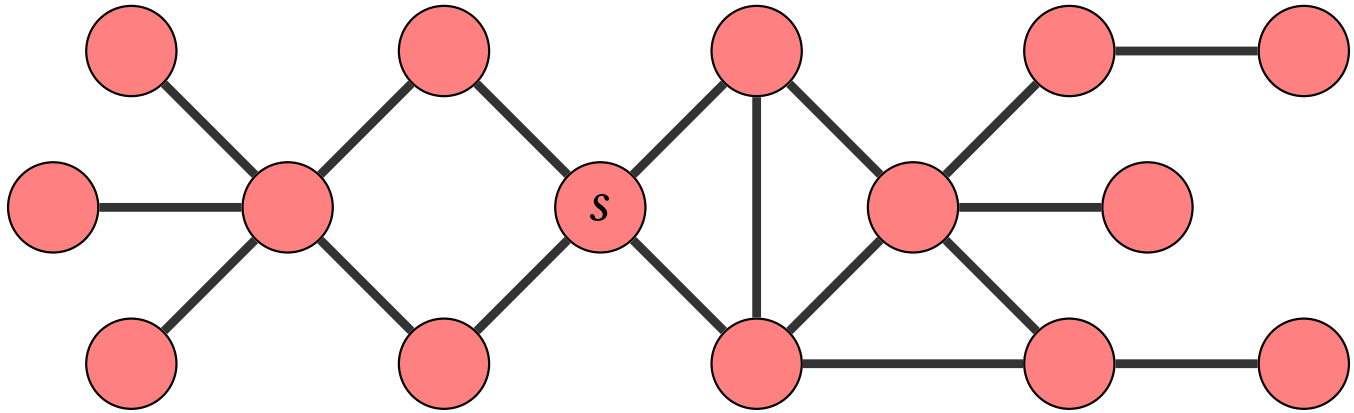


Broadcast Algorithm

Initial situation: Node s has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors



Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\mathrm{dist}(s, v)$ (*distance* from s to v): length of a shortest path from s to v

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\mathrm{dist}(s, v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s, v) \geq 1$

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\mathrm{dist}(s, v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s, v) \geq 1$

- **Base case** ($\mathrm{dist}(s, v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s, v) + 1$
- **Inductive step** ($\mathrm{dist}(s, v) \geq 2$):

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\mathrm{dist}(s, v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s, v) \geq 1$

- **Base case** ($\mathrm{dist}(s, v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s, v) + 1$
- **Inductive step** ($\mathrm{dist}(s, v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\mathrm{dist}(s, v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s, v) \geq 1$

- **Base case** ($\mathrm{dist}(s, v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s, v) + 1$
- **Inductive step** ($\mathrm{dist}(s, v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s, u) = \mathrm{dist}(s, v) - 1$

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$
- **Inductive step** ($\mathrm{dist}(s,v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$
  - By the induction hypothesis: u received information I by round $\mathrm{dist}(s,u) + 1$ at the latest and sent it to all neighbors

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$
- **Inductive step** ($\mathrm{dist}(s,v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$
  - By the induction hypothesis: u received information I by round $\mathrm{dist}(s,u) + 1$ at the latest and sent it to all neighbors
  - Since v is a neighbor of u, v receives information I by round $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$ at the latest

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$
- **Inductive step** ($\mathrm{dist}(s,v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$
  - By the induction hypothesis: u received information I by round $\mathrm{dist}(s,u) + 1$ at the latest and sent it to all neighbors
  - Since v is a neighbor of u, v receives information I by round $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$ at the latest
- **Correctness**:

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$
- **Inductive step** ($\mathrm{dist}(s,v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$
  - By the induction hypothesis: u received information I by round $\mathrm{dist}(s,u) + 1$ at the latest and sent it to all neighbors
  - Since v is a neighbor of u, v receives information I by round $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$ at the latest
- **Correctness**:
  - Since the network is connected, $\mathrm{dist}(s,v)$ is *finite* for every node v

Correctness

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s,v) + 1$ at the latest.

$\mathrm{dist}(s,v)$ (*distance* from s to v): length of a shortest path from s to v

Observation: For $v \neq s$, $\mathrm{dist}(s,v) \geq 1$

- **Base case** ($\mathrm{dist}(s,v) = 1$): v receives I from s in round $2 = \mathrm{dist}(s,v) + 1$
- **Inductive step** ($\mathrm{dist}(s,v) \geq 2$):
  - Let u be a predecessor of v on a shortest path from s to v
  - We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) - 1$
  - By the induction hypothesis: u received information I by round $\mathrm{dist}(s,u) + 1$ at the latest and sent it to all neighbors
  - Since v is a neighbor of u, v receives information I by round $\mathrm{dist}(s,u) + 2 = \mathrm{dist}(s,v) + 1$ at the latest
- **Correctness**:
  - Since the network is connected, $\mathrm{dist}(s,v)$ is *finite* for every node v
  - Thus: after finitely many rounds, every node knows information I

Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is



Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u, v)$.



Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u, v)$.

Definition

The **diameter** of the network is $D = \max_{u,v} \mathrm{dist}(u, v)$.



Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u, v)$.

Definition

The **diameter** of the network is $D = \max_{u,v} \mathrm{dist}(u, v)$.

$D = \max_u \mathrm{Ecc}(u)$



Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u, v)$.

Definition

The **diameter** of the network is $D = \max_{u,v} \mathrm{dist}(u, v)$.

$D = \max_u \mathrm{Ecc}(u)$, hence: $\mathrm{Ecc}(s) \leq D$



Running Time

Induction claim

For every node $v \neq s$: node v receives information I by round $\mathrm{dist}(s, v) + 1$ at the latest.

$\Rightarrow$ number of rounds: $\max_v \mathrm{dist}(s, v) + 1$

Definition

The **eccentricity** of a node u is $\mathrm{Ecc}(u) = \max_v \mathrm{dist}(u, v)$.

Definition

The **diameter** of the network is $D = \max_{u,v} \mathrm{dist}(u, v)$.



$D = \max_u \mathrm{Ecc}(u)$, hence: $\mathrm{Ecc}(s) \leq D$

Running time of the broadcast algorithm: $O(\mathrm{Ecc}(s)) = O(D) = O(n)$

Message Complexity

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

Message Complexity

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

Analysis:

- Every node forwards information I to all neighbors *once*

Message Complexity

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

Analysis:

- Every node forwards information I to all neighbors *once*
- $\deg(v)$ messages per node v

Message Complexity

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all neighbors

Analysis:

- Every node forwards information I to all neighbors *once*
- $\deg(v)$ messages per node v
- In total: $\sum_v \deg(v) = 2m = O(m)$ messages

Breadth-First Search Tree



Breadth-First Search (BFS) Tree

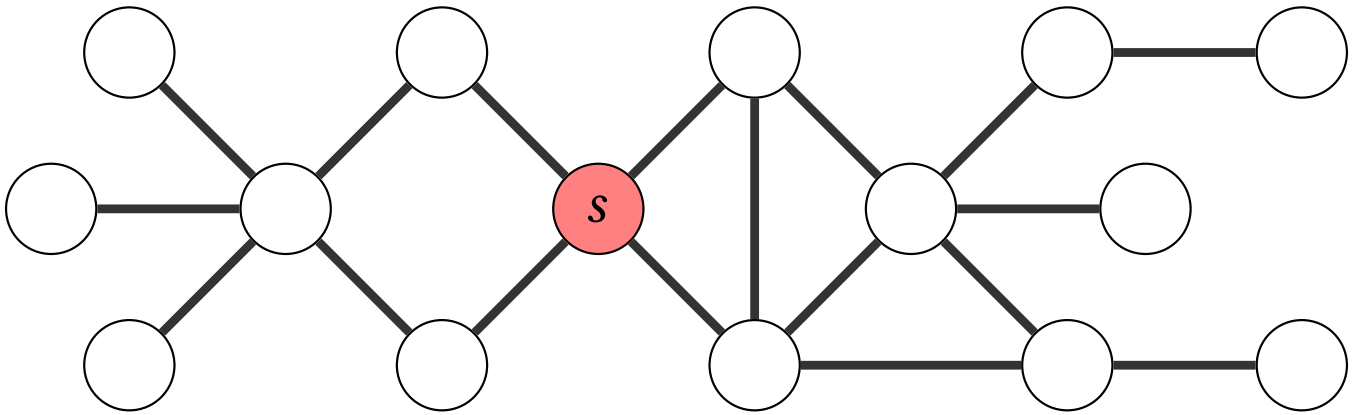
Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors

Breadth-First Search (BFS) Tree

Algorithm:

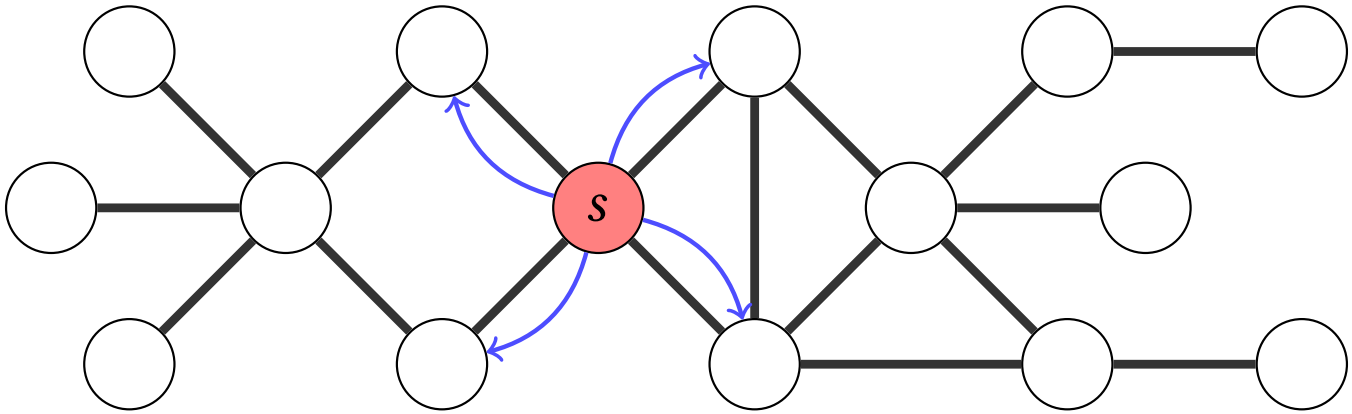
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

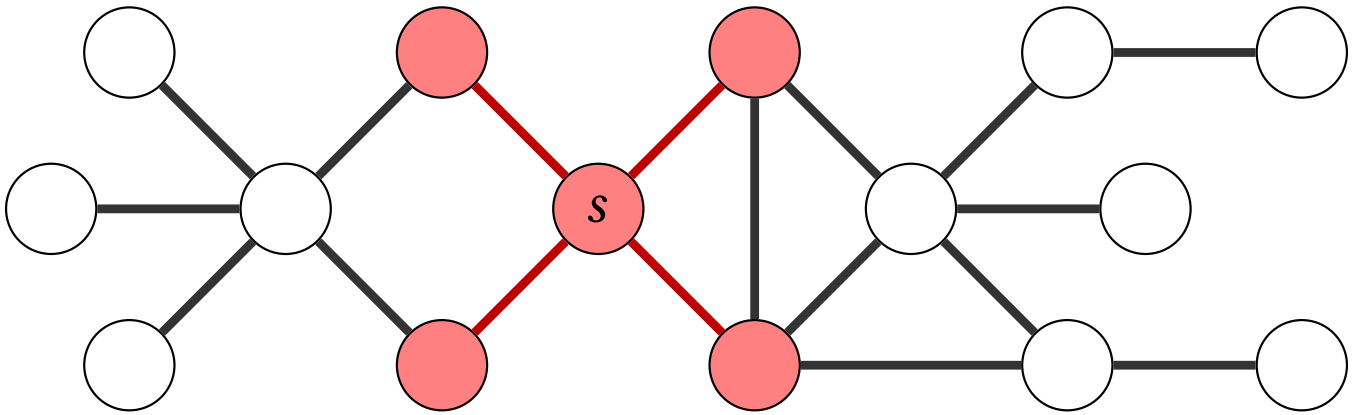
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

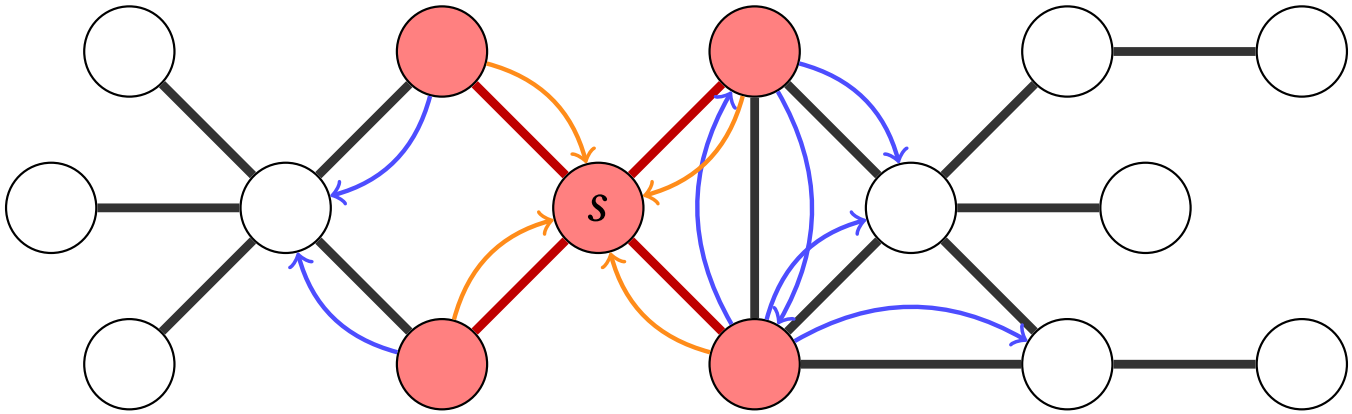
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

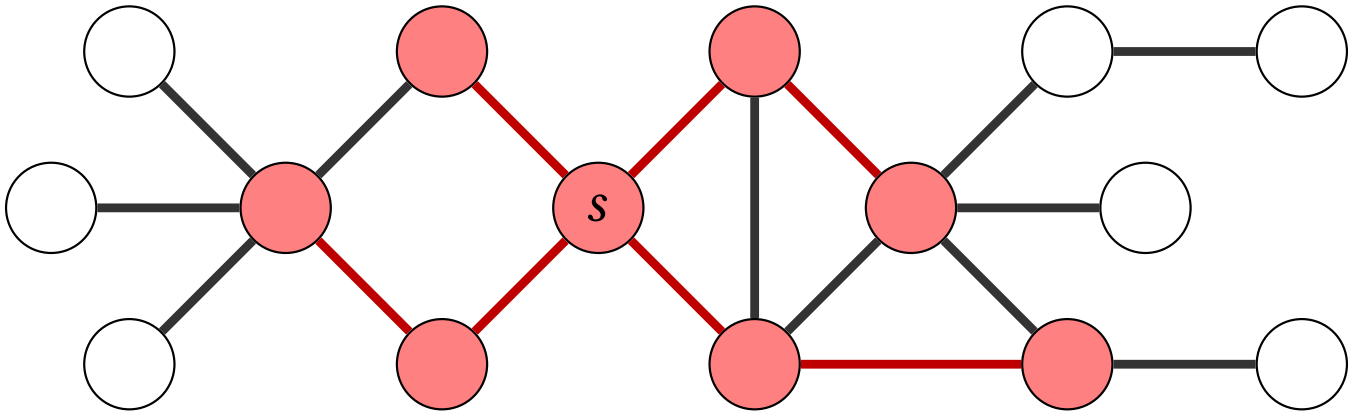
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

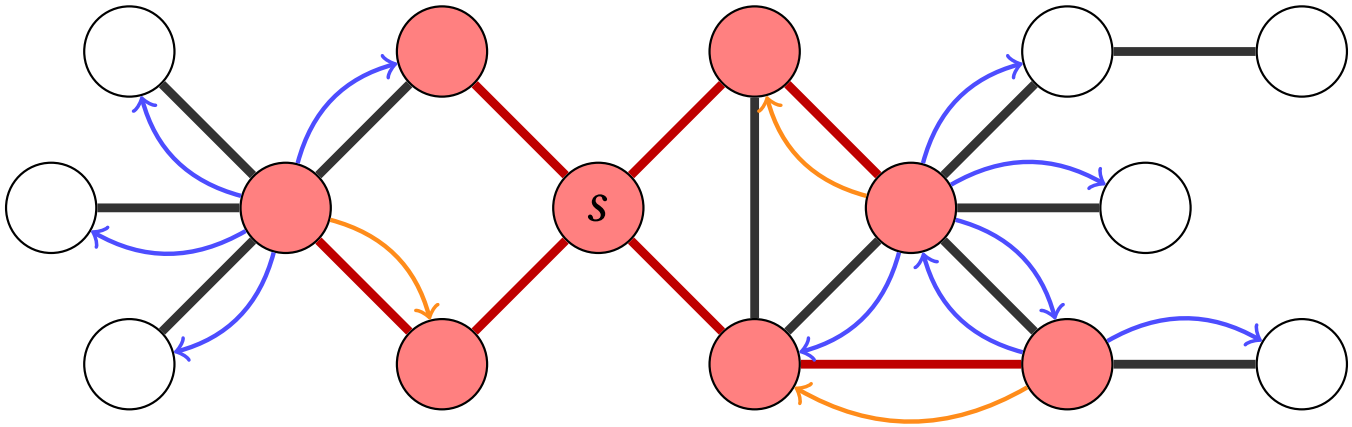
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

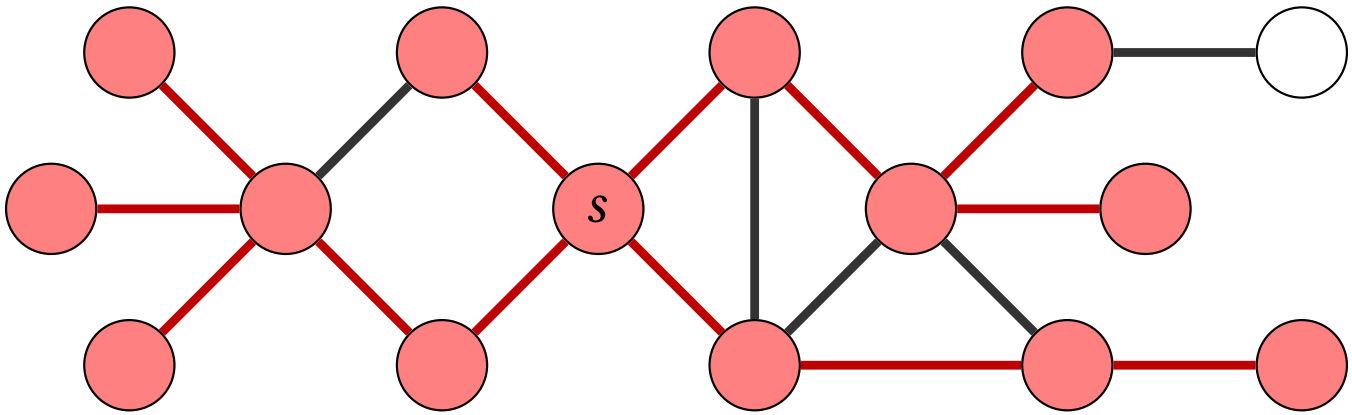
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

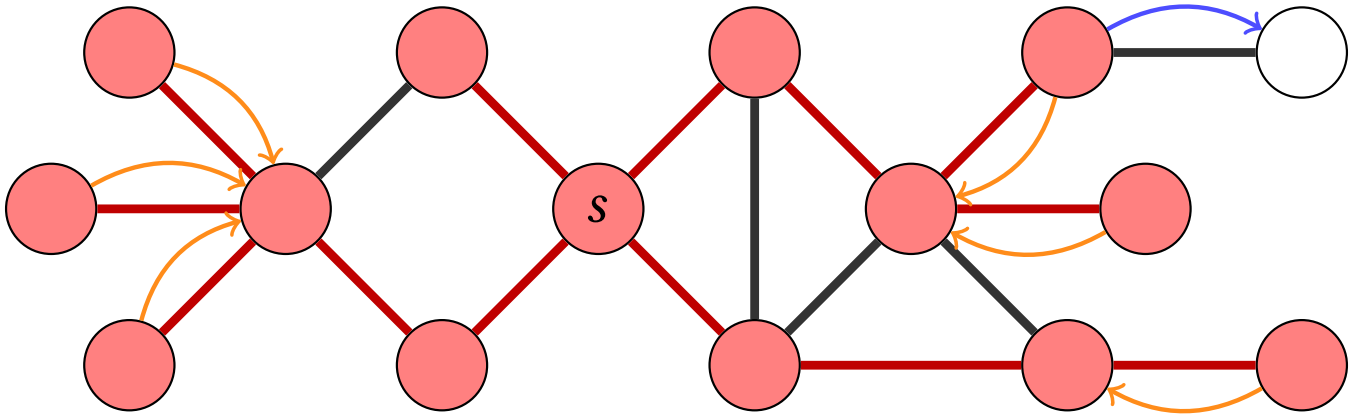
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

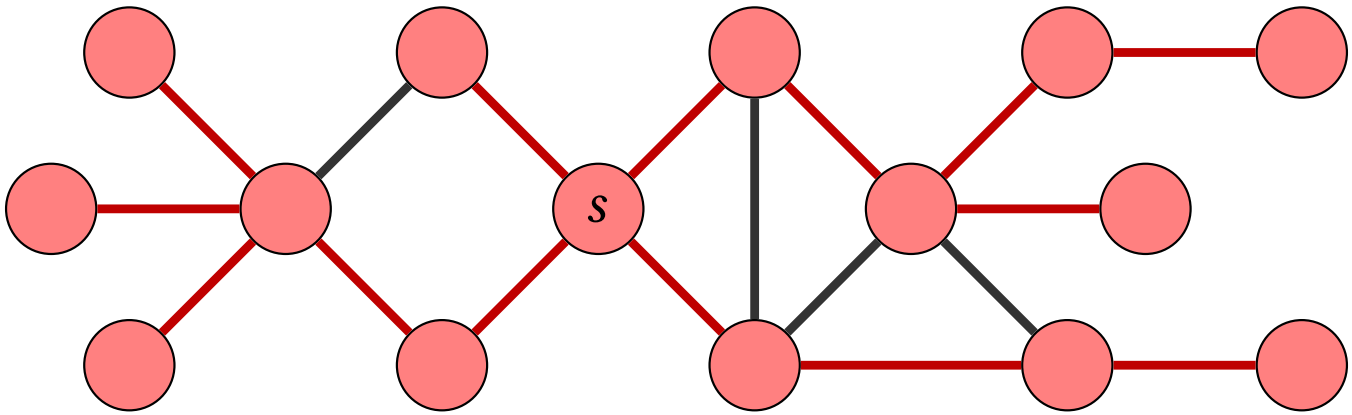
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

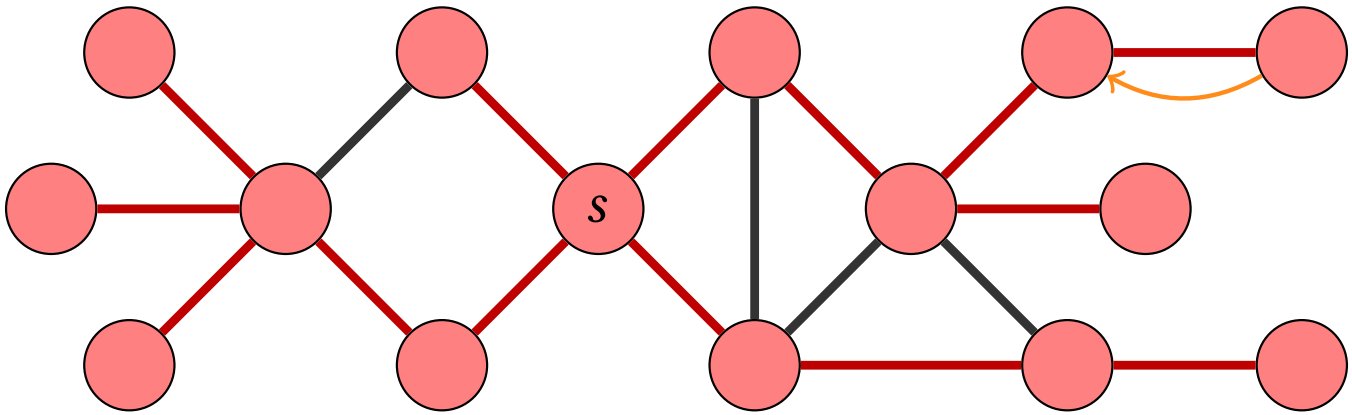
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

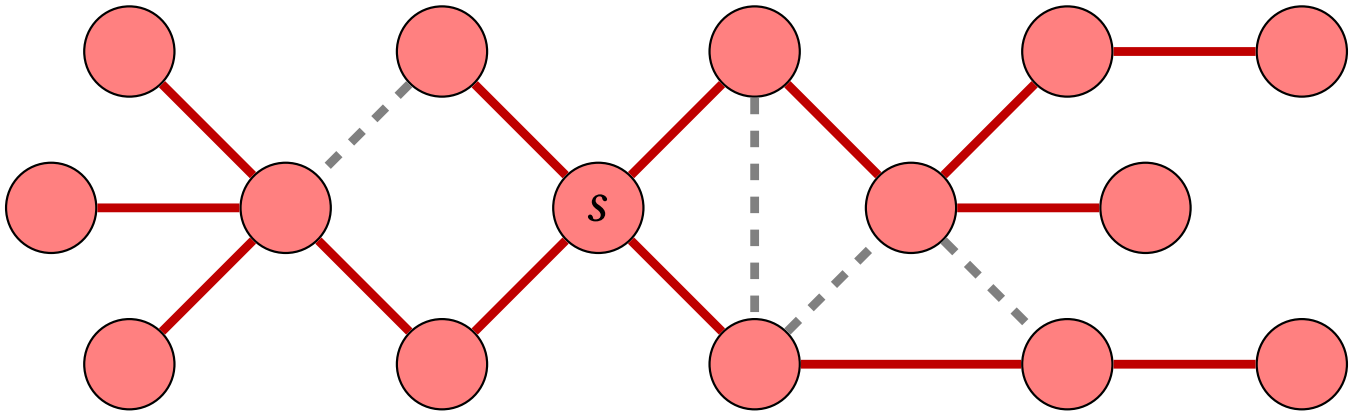
- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Breadth-First Search (BFS) Tree

Algorithm:

- In round 1, s sends information I to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time:
  - v stores *one* of the senders of I as its parent
    (Tie-breaking: for example, prefer the node with the smaller ID)
  - v sends a "child" message to its parent and I to all other neighbors



Implicit spanning tree: Every node knows its parent and children

Distance Computation

Idea: Breadth-first search induces distances from s

Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1

Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent

Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

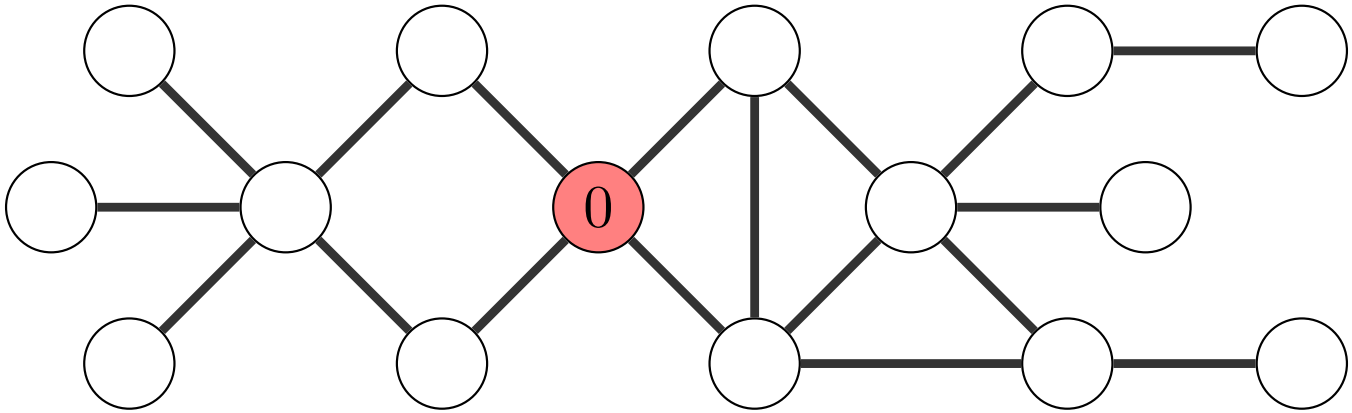
- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

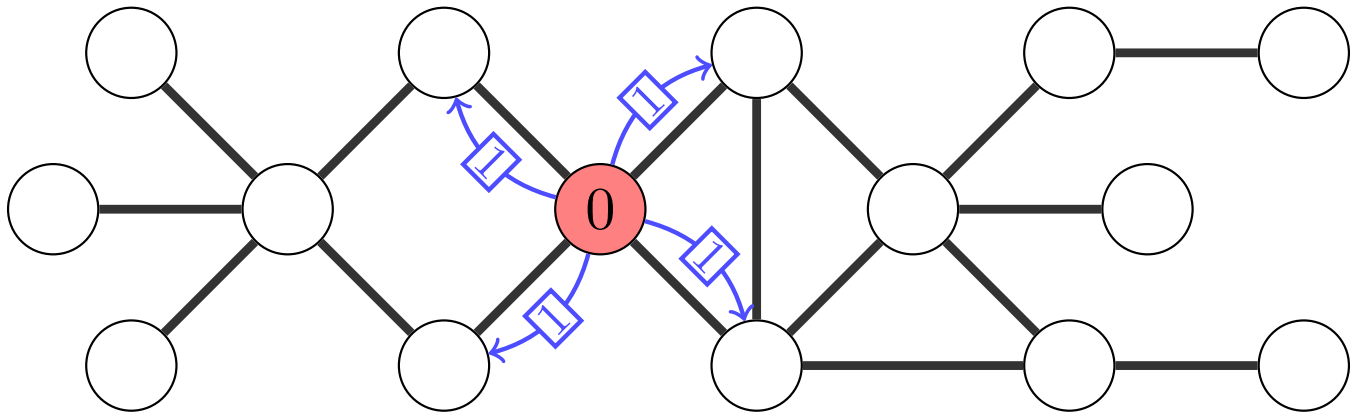


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

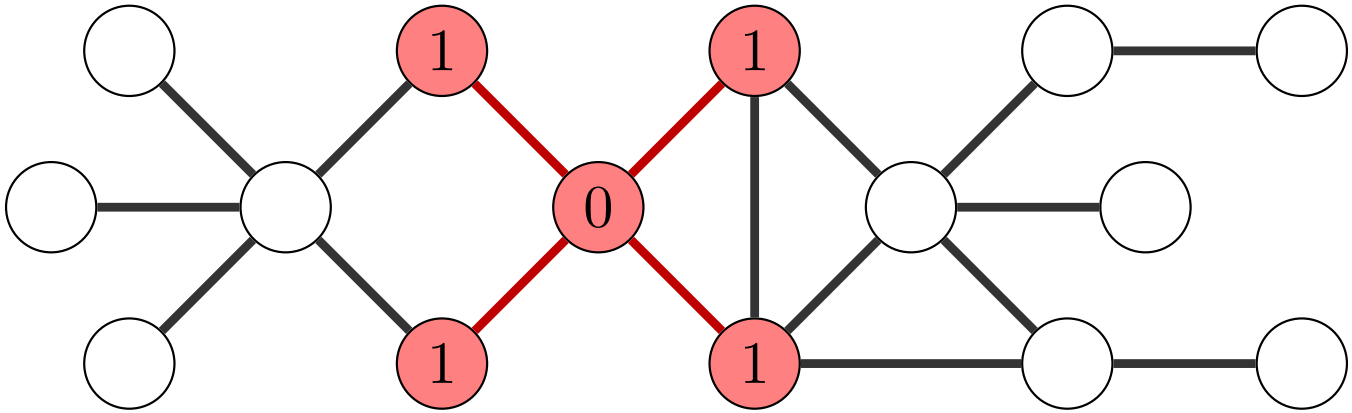


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

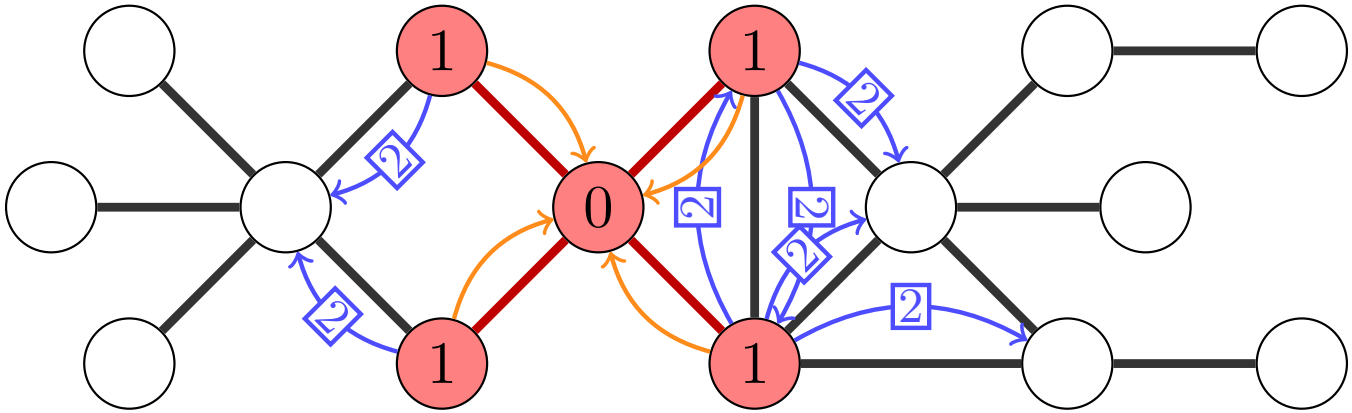


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

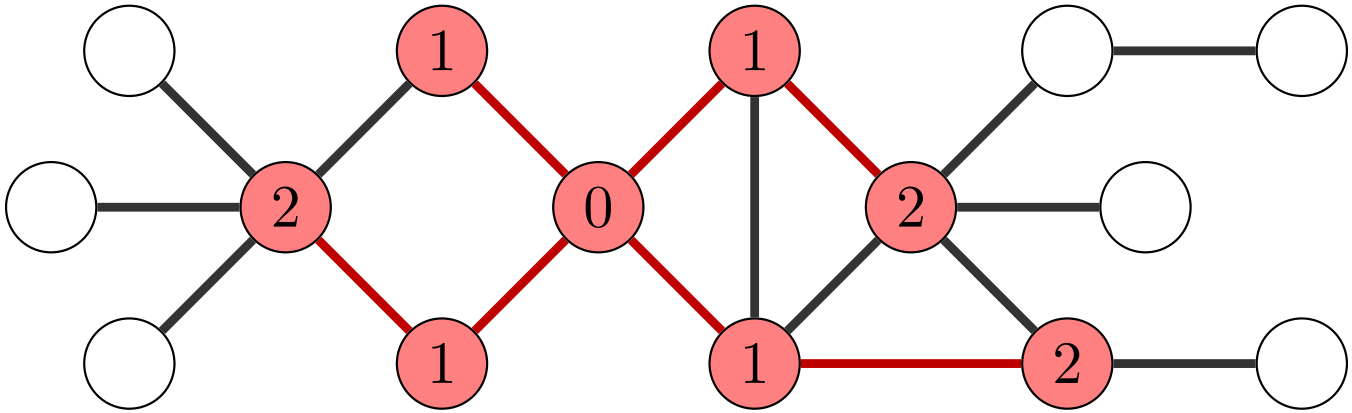


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

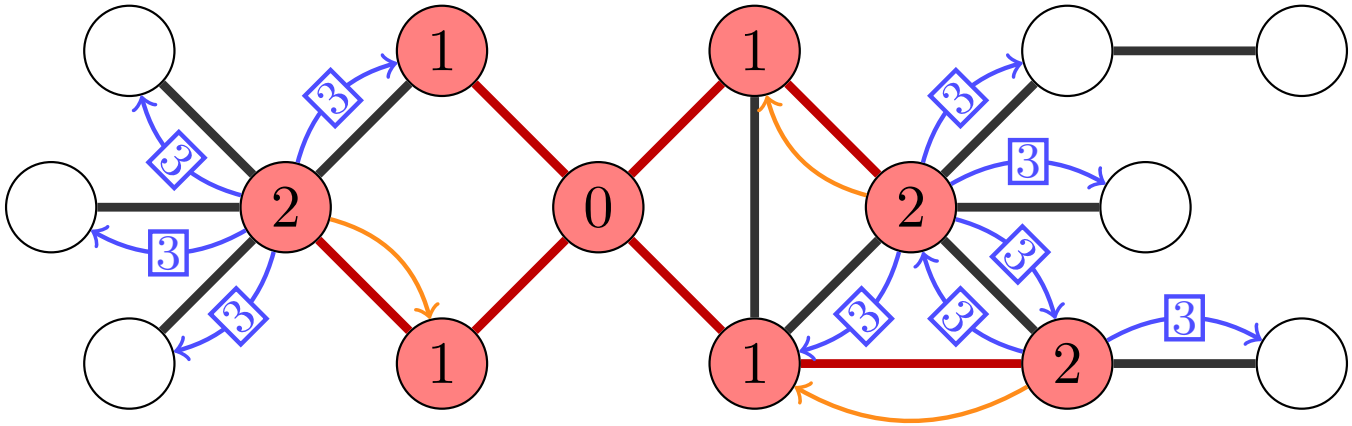


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

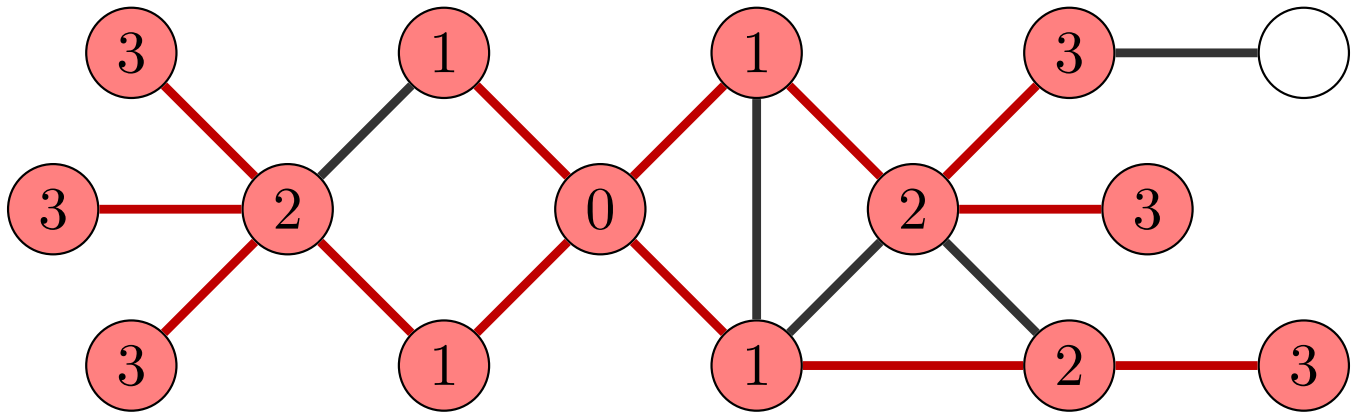


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

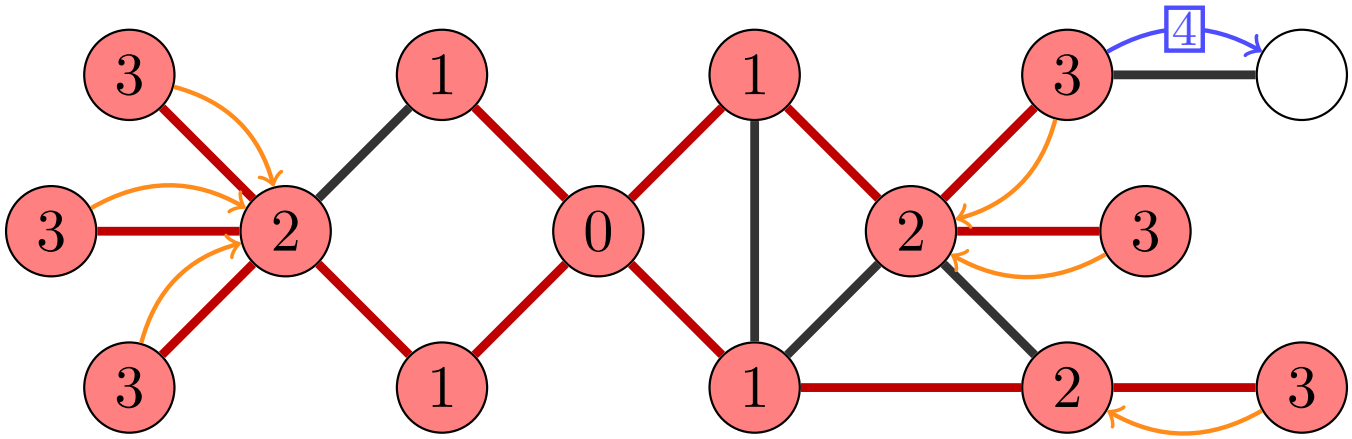


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

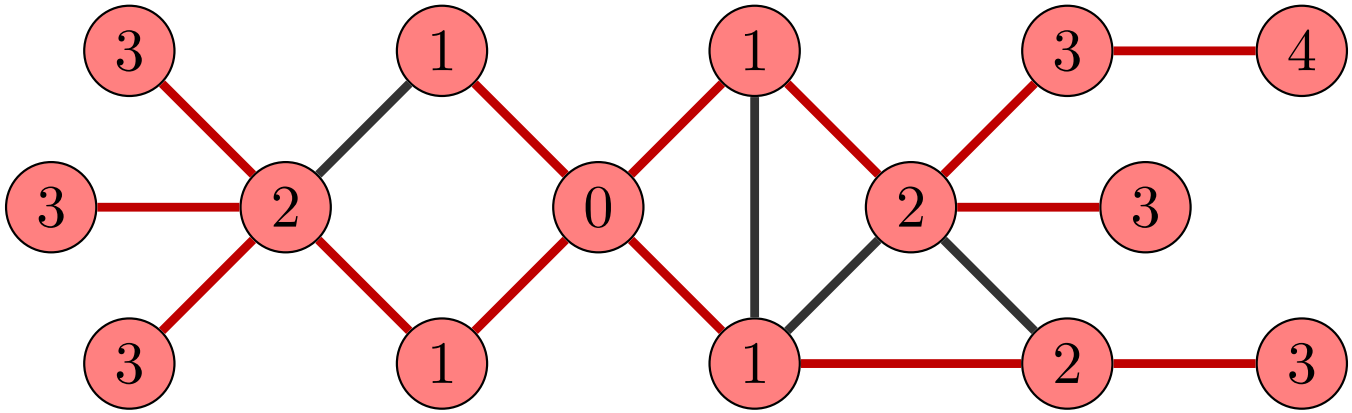


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

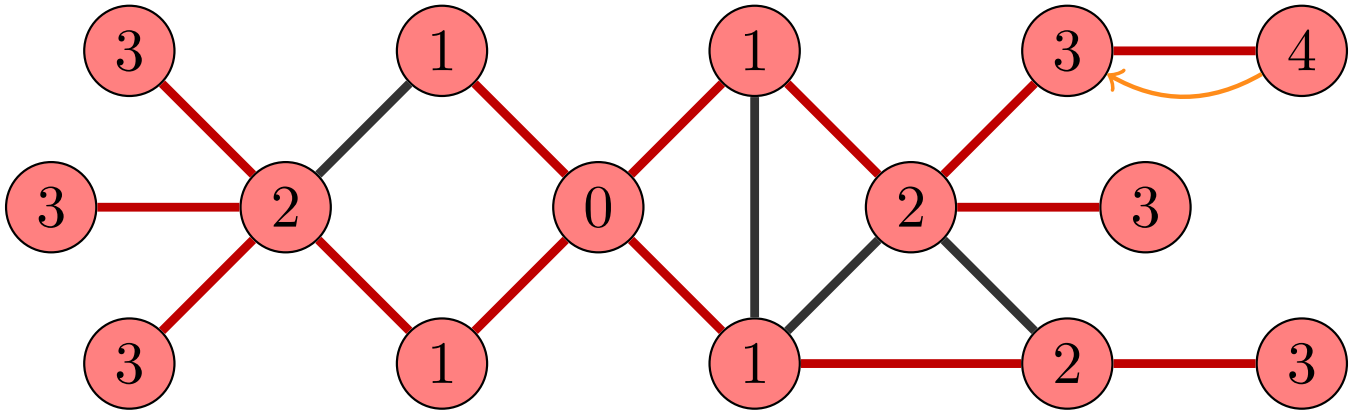


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors

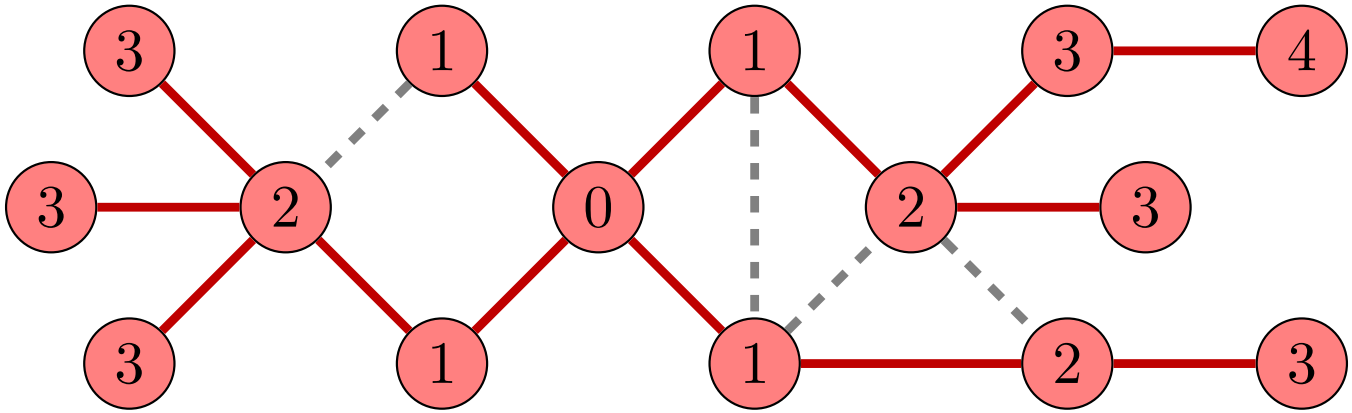


Distance Computation

Idea: Breadth-first search induces distances from s

Algorithm:

- s has distance 0 from itself and sends distance information 1 to all neighbors in round 1
- Whenever some node $v \neq s$ receives distance information d for the first time:
  - v stores one of the senders of d as its parent
  - v sends a "child" message to its parent and distance information $d + 1$ to all other neighbors



Breadth-First Search Summary

Breadth-first search with start node s:

- #rounds: $O(\mathrm{Ecc}(s)) = O(D)$
- #messages: $O(m)$

Breadth-First Search Summary

Breadth-first search with start node s:

- #rounds: $O(\mathrm{Ecc}(s)) = O(D)$
- #messages: $O(m)$
- Every node v knows the distance $\mathrm{dist}(s, v)$

Breadth-First Search Summary

Breadth-first search with start node s:

- #rounds: $O(\mathrm{Ecc}(s)) = O(D)$
- #messages: $O(m)$
- Every node v knows the distance $\mathrm{dist}(s, v)$
- Breadth-first search tree:
  - Every node $u \neq s$ has a parent v with $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) - 1$

Breadth-First Search Summary

Breadth-first search with start node s:

- #rounds: $O(\mathrm{Ecc}(s)) = O(D)$
- #messages: $O(m)$
- Every node v knows the distance $\mathrm{dist}(s, v)$
- Breadth-first search tree:
  - Every node $u \neq s$ has a parent v with $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) - 1$
  - Every node knows its parent and children in the tree

Breadth-First Search Summary

Breadth-first search with start node s:

- #rounds: $O(\mathrm{Ecc}(s)) = O(D)$
- #messages: $O(m)$
- Every node v knows the distance $\mathrm{dist}(s, v)$
- Breadth-first search tree:
  - Every node $u \neq s$ has a parent v with $\mathrm{dist}(s, v) = \mathrm{dist}(s, u) - 1$
  - Every node knows its parent and children in the tree
  - Every BFS tree is also a spanning tree

Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children

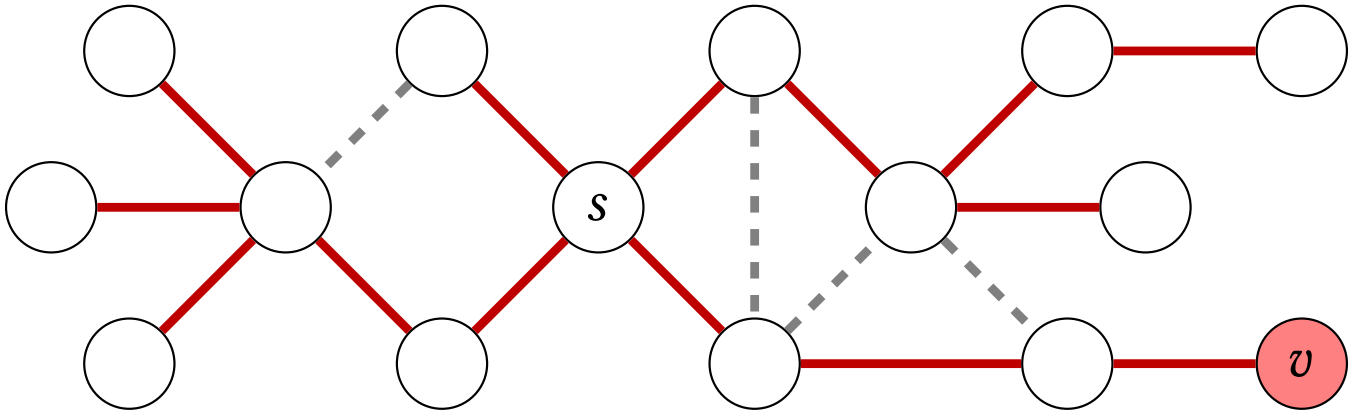
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



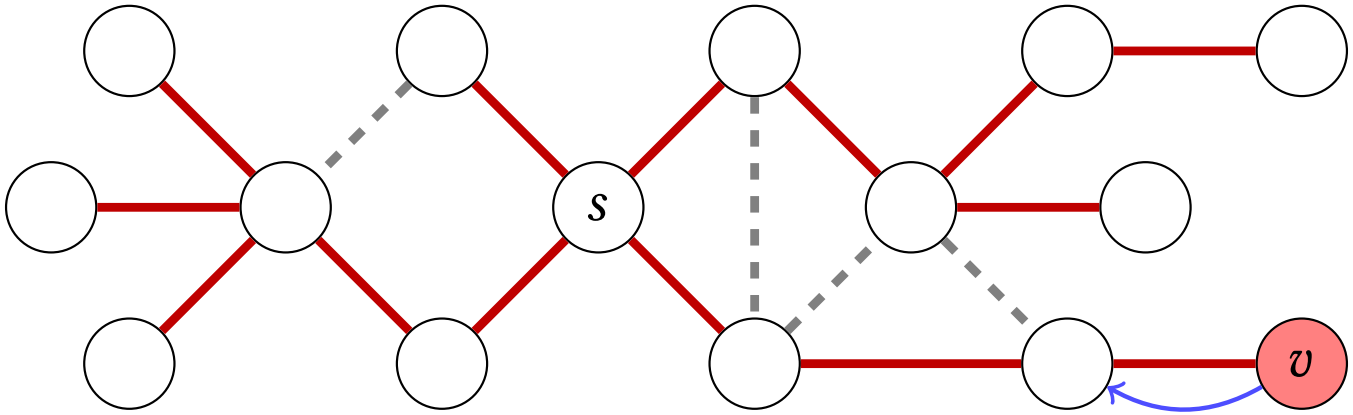
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



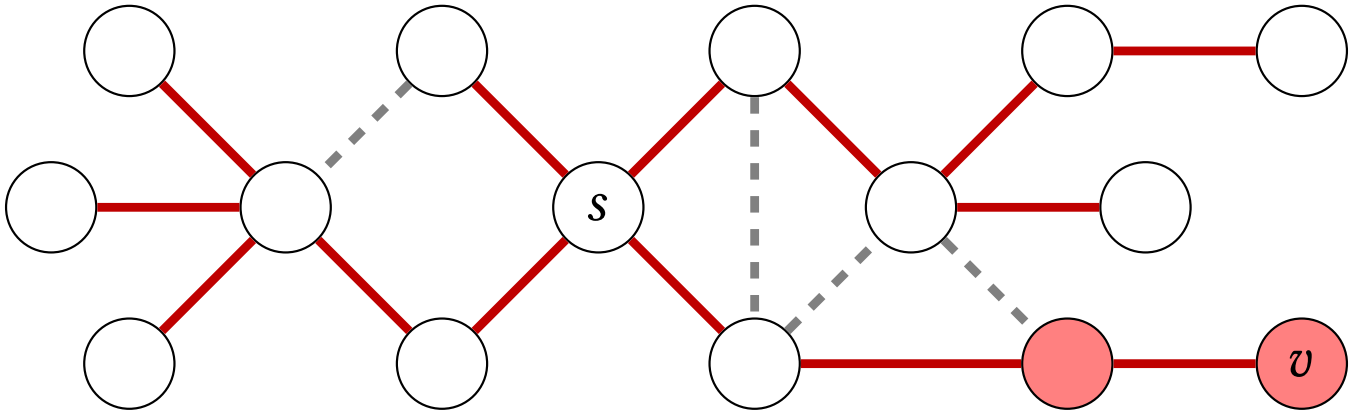
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



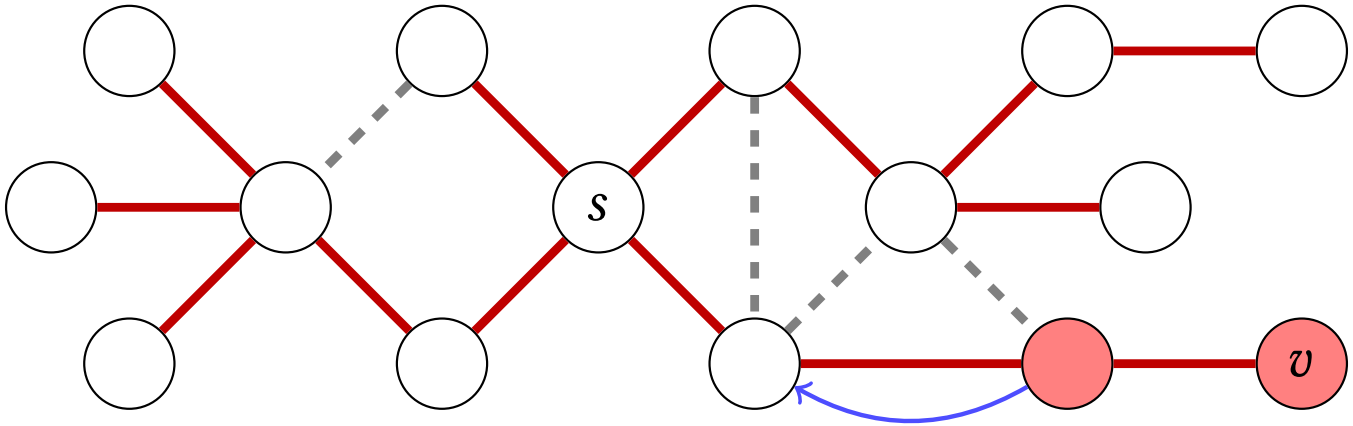
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



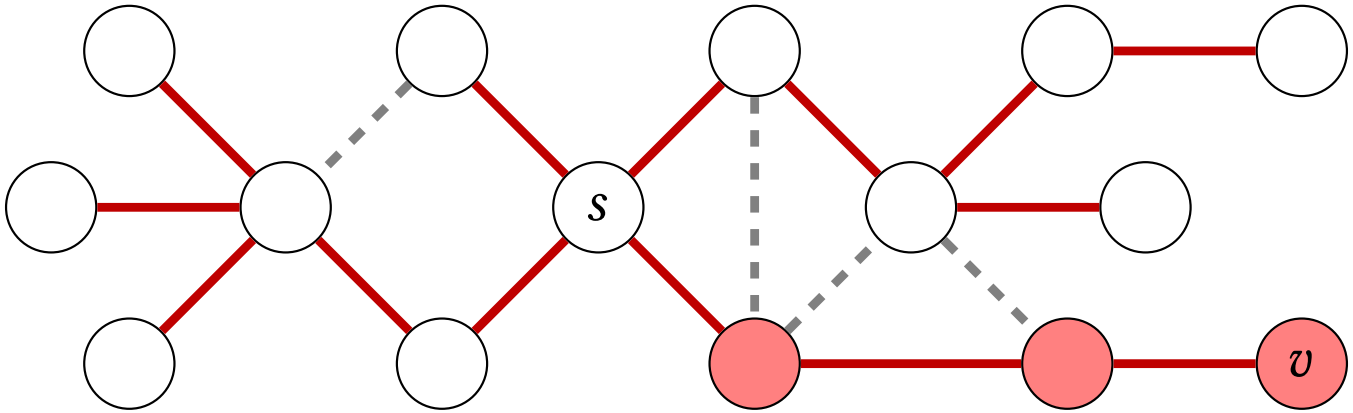
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



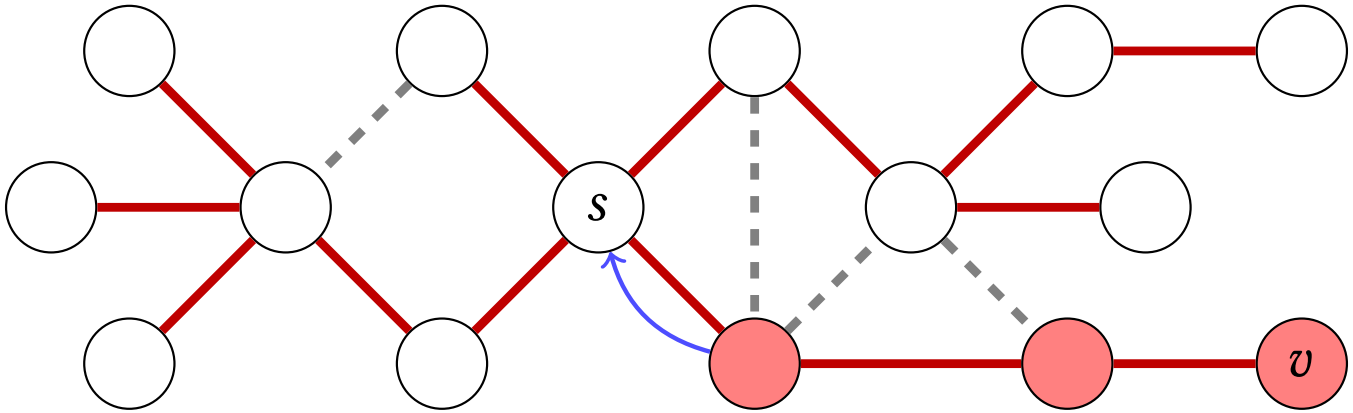
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



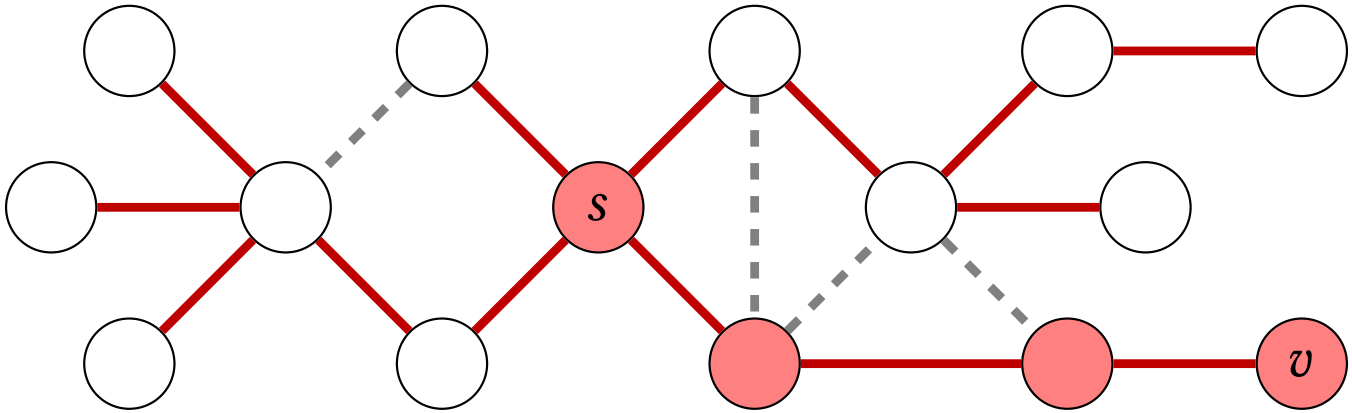
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



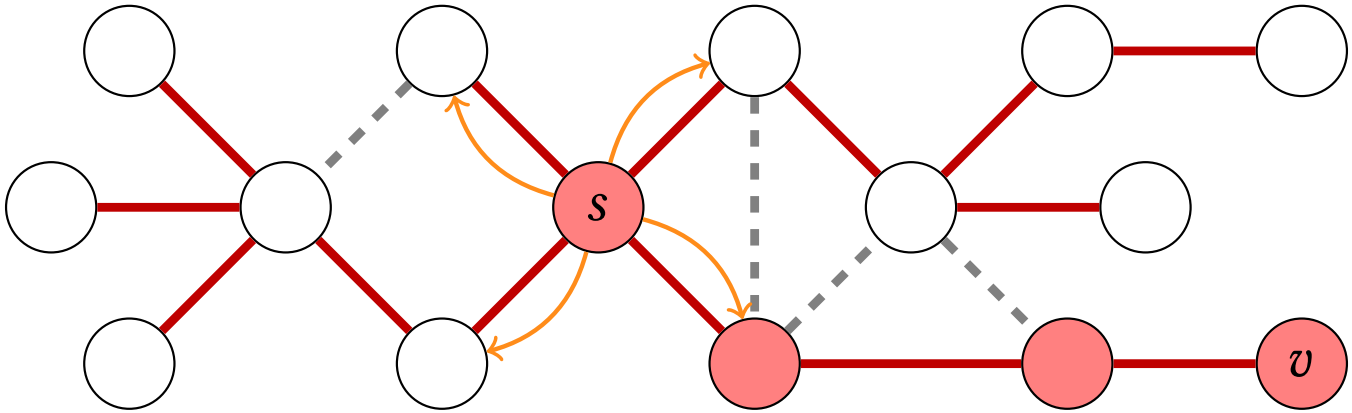
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



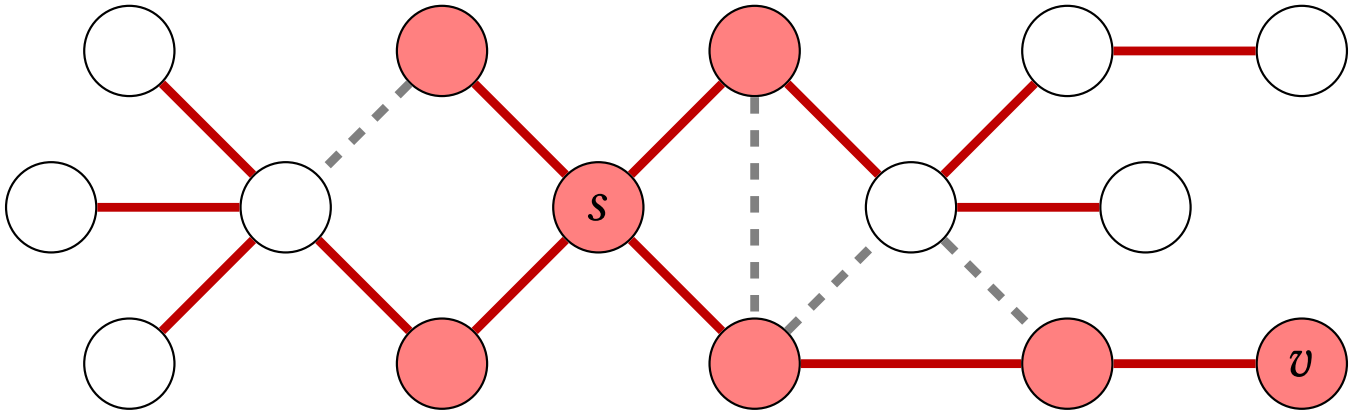
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



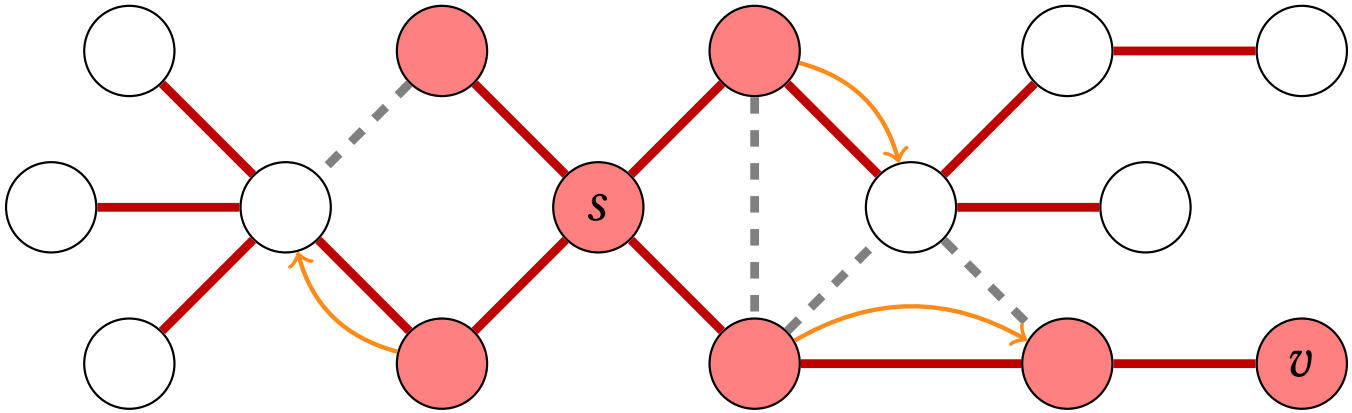
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



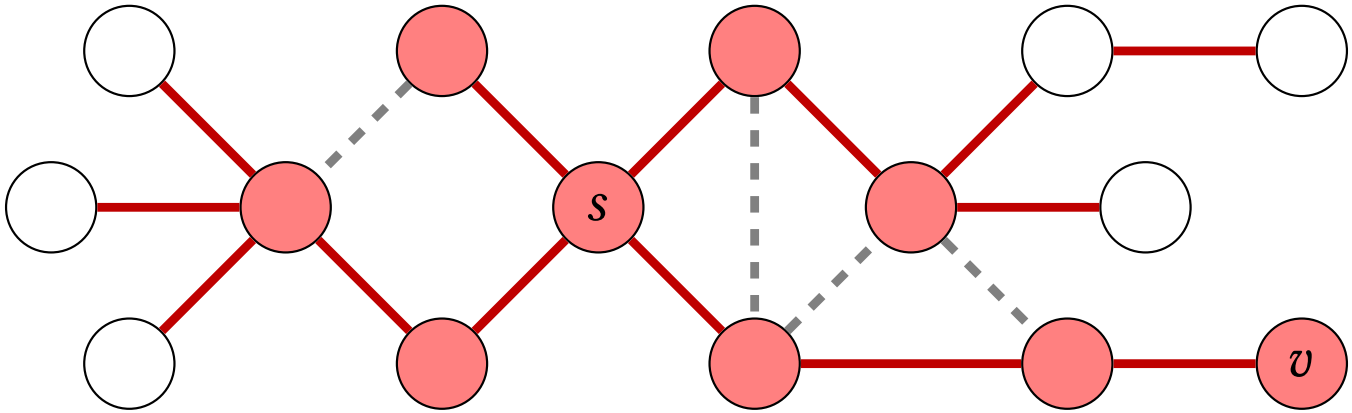
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



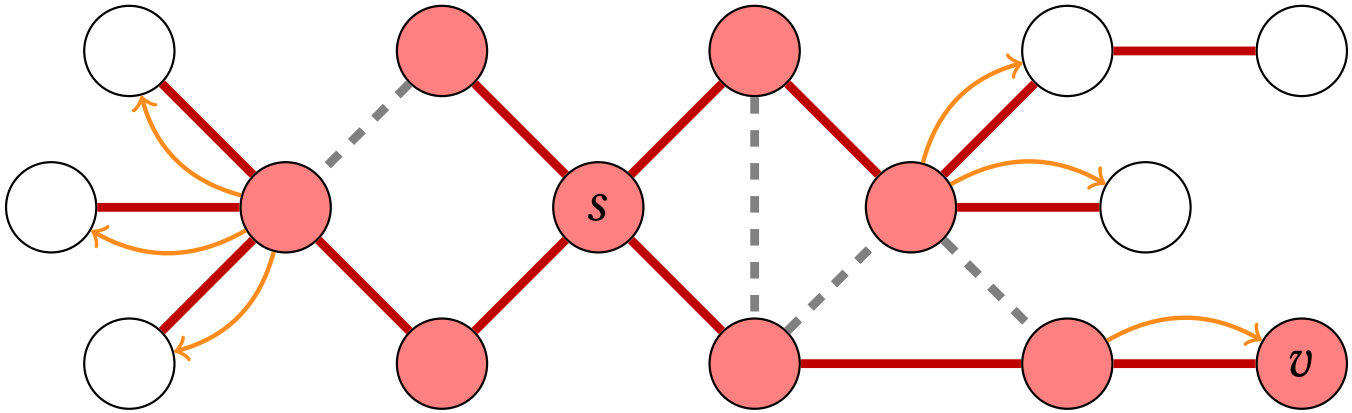
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



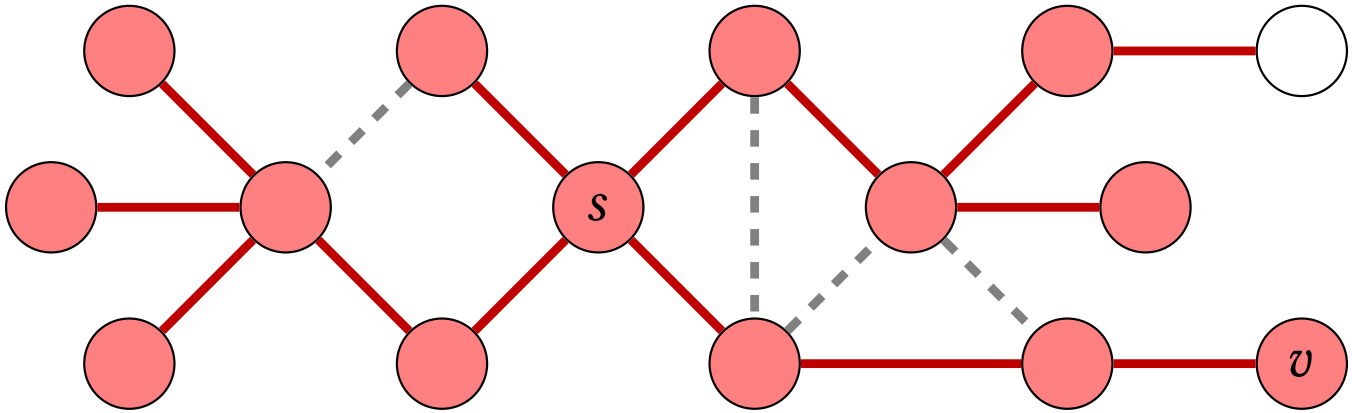
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



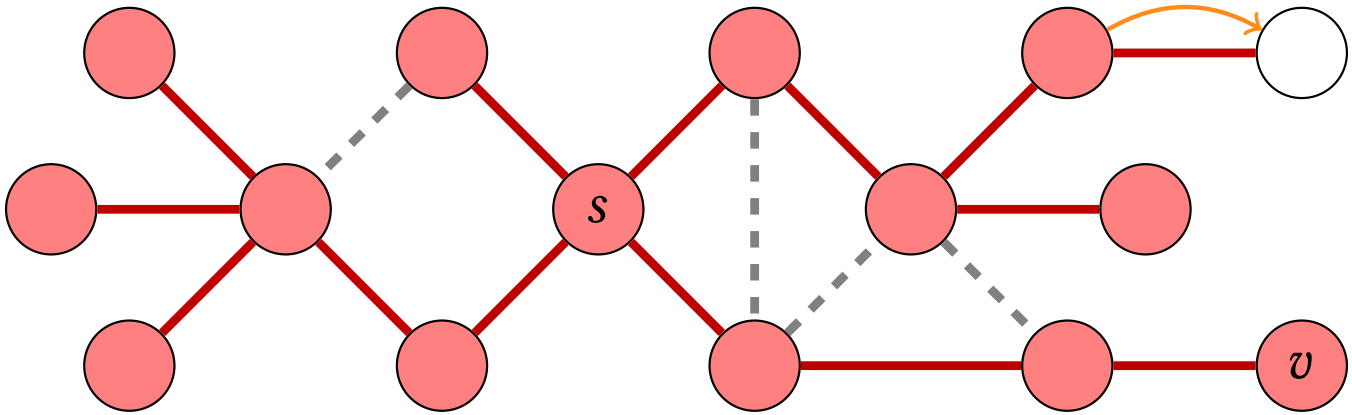
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



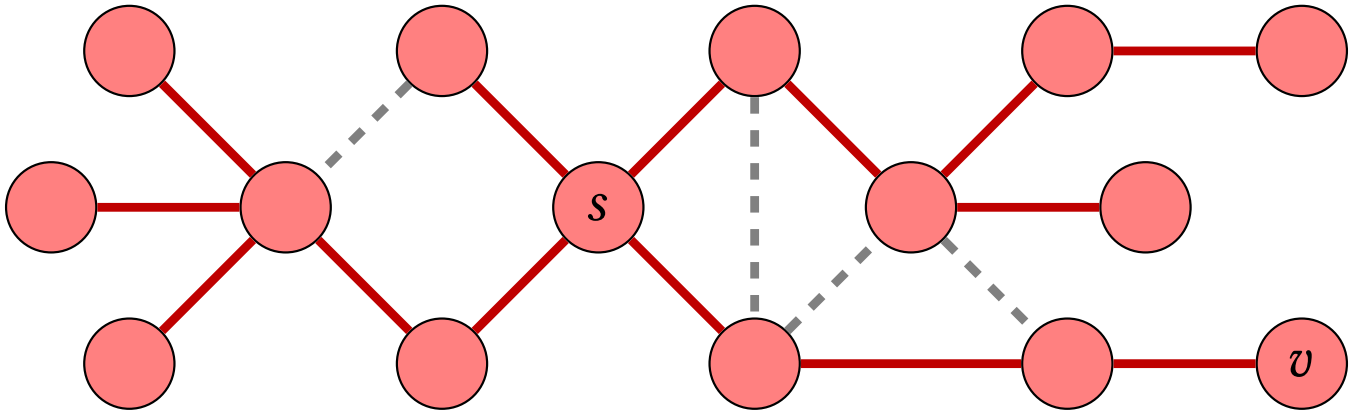
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



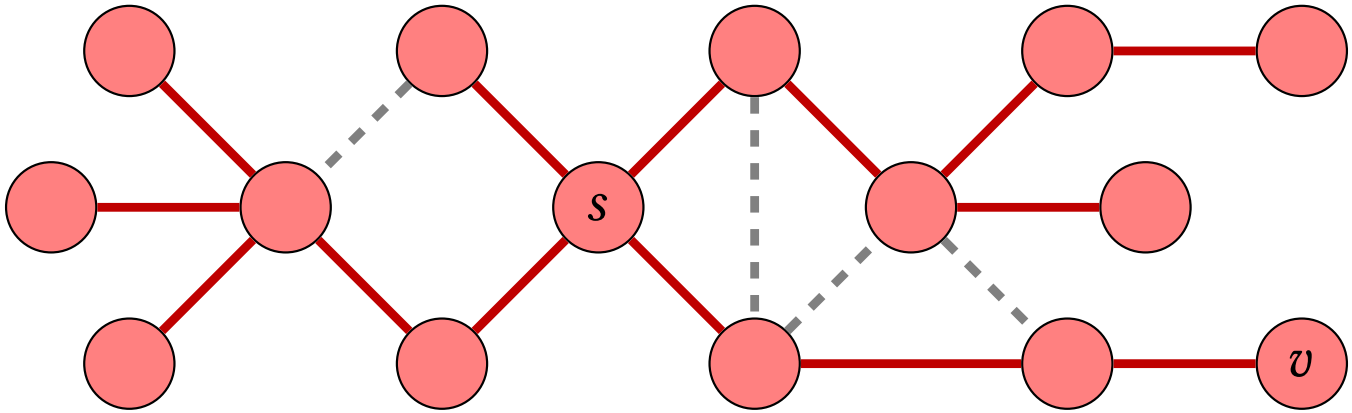
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



#rounds: $O(D)$

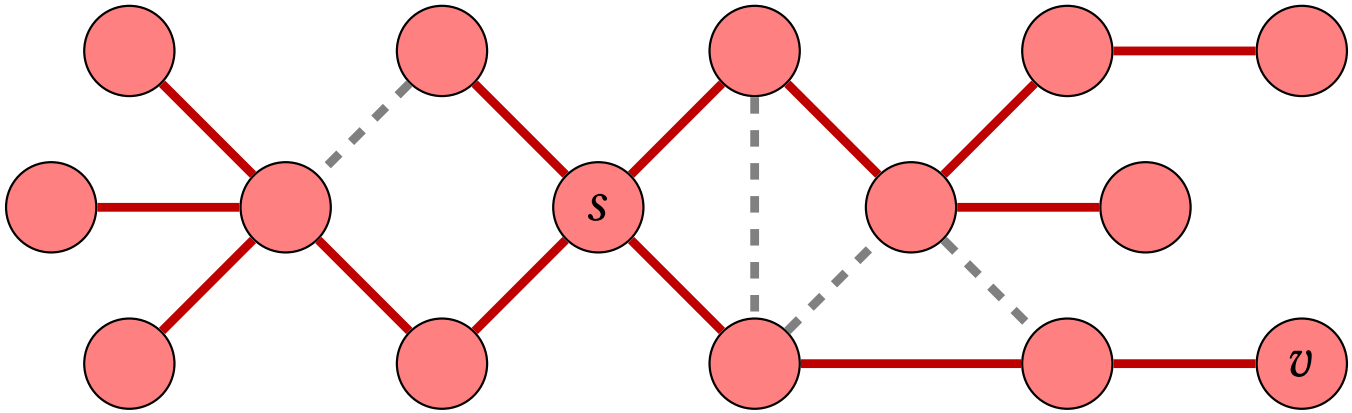
Upcast+Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node v has information I (of size $O(\log n)$)
Goal: All nodes have information I

Algorithm:

- Upcast: I is sent to s via parent nodes
- Downcast: I is sent from s to all nodes via the children



#rounds: $O(D)$
#messages: $O(n)$ (instead of $O(m)$ as in broadcast)

Pipelining



Multiple Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node s has k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: All nodes have information $I_1, \ldots, I_k$

Multiple Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node s has k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: All nodes have information $I_1, \ldots, I_k$

Caution: The limited message size of $O(\log n)$ prevents bundling them into a single message!

Multiple Downcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: Node s has k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: All nodes have information $I_1, \ldots, I_k$

Caution: The limited message size of $O(\log n)$ prevents bundling them into a single message!

Naive solution:

- Use the downcast algorithm k times
- Running time: $O(kD)$

Pipeline Algorithm

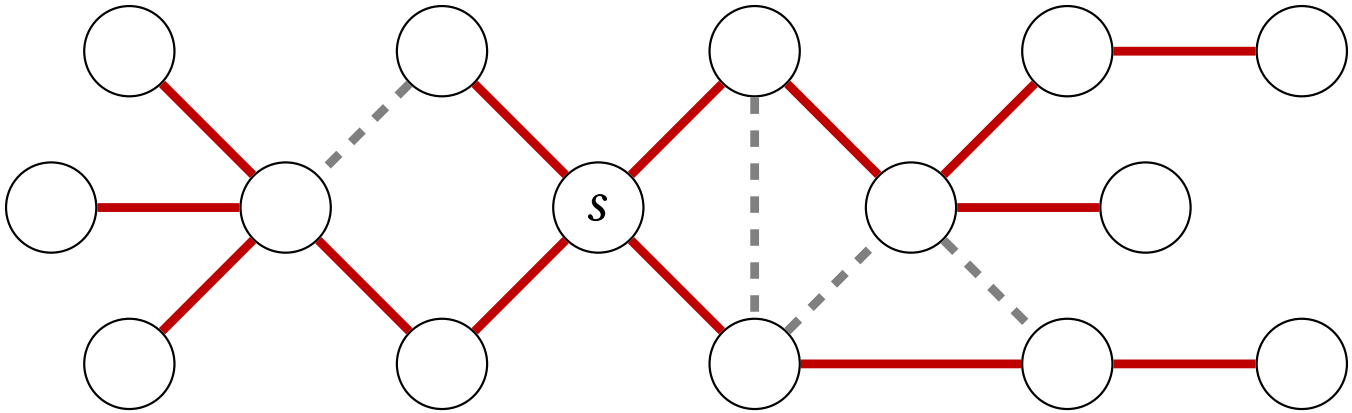
Algorithm:

- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children

Pipeline Algorithm

Algorithm:

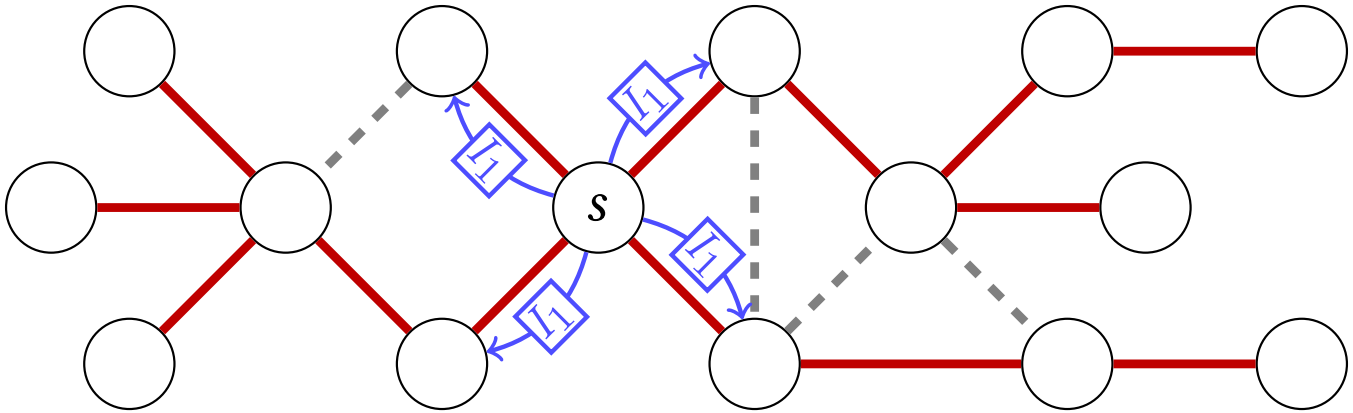
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

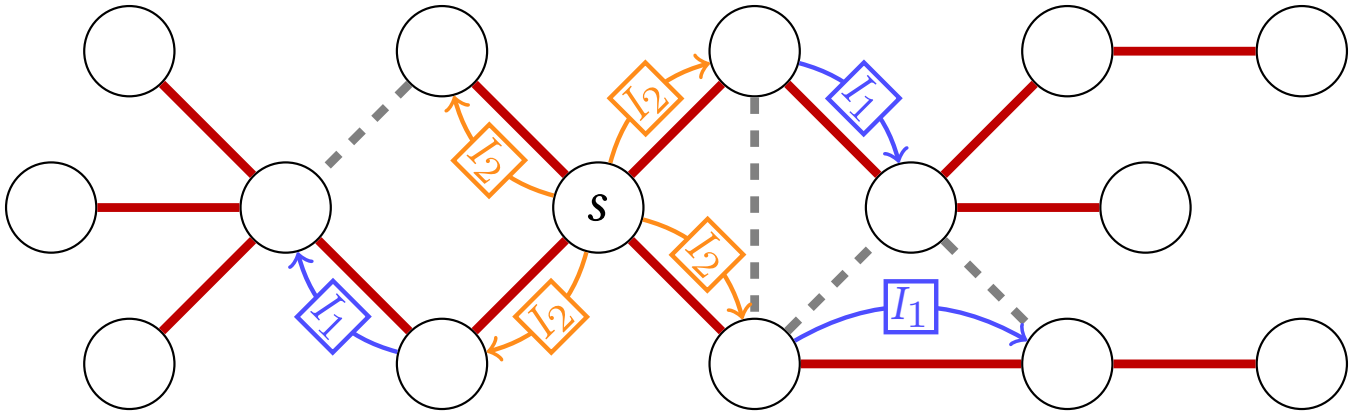
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

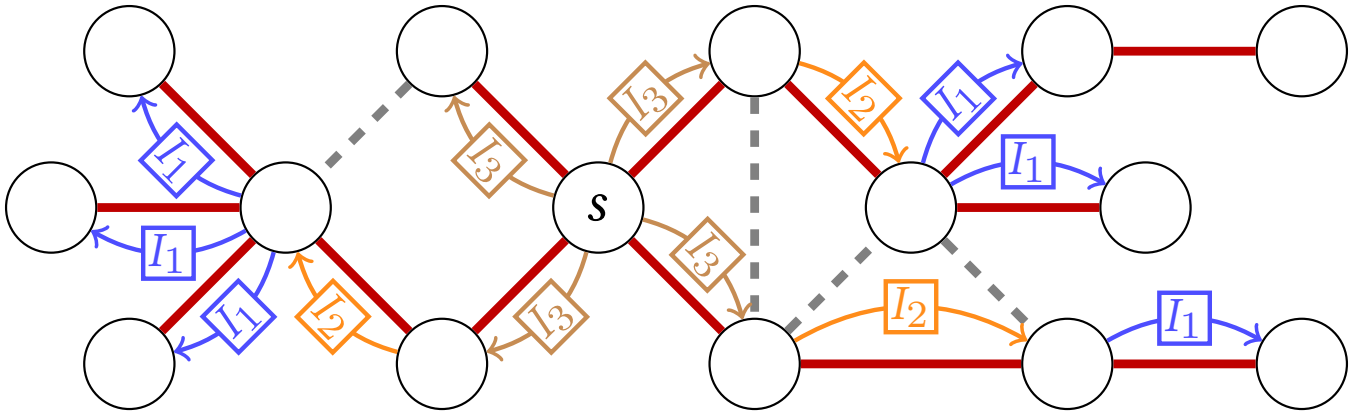
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

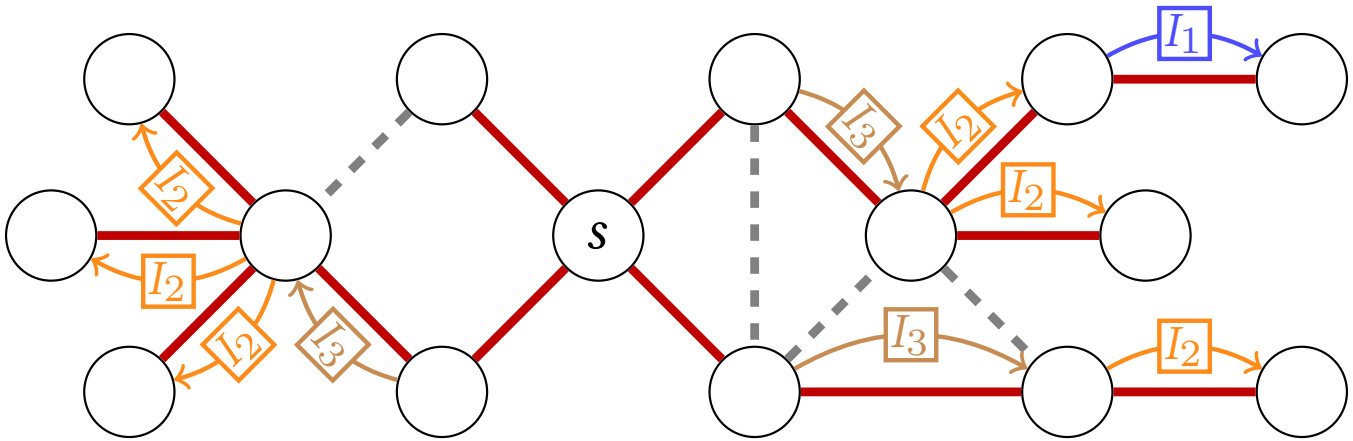
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

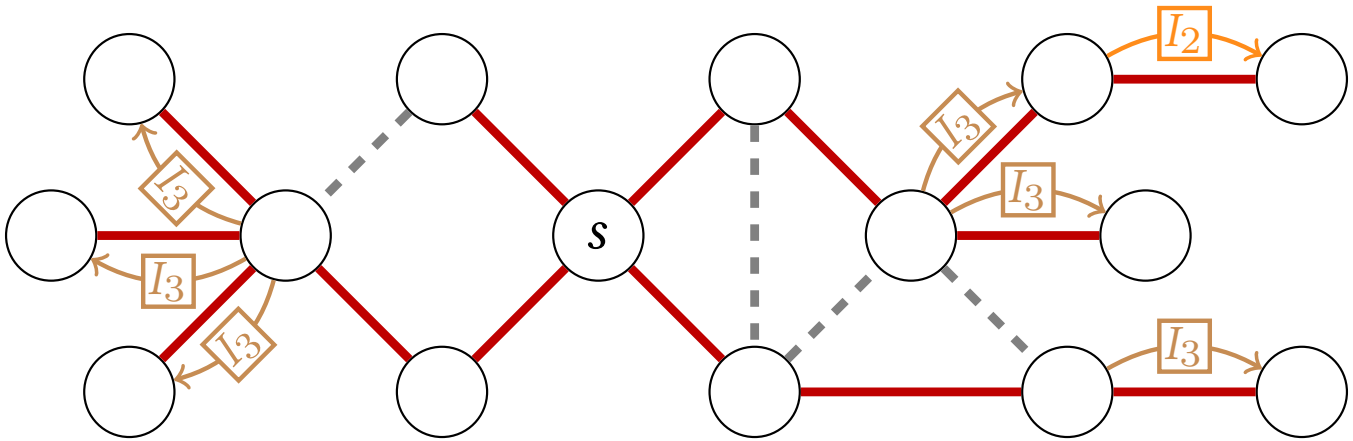
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

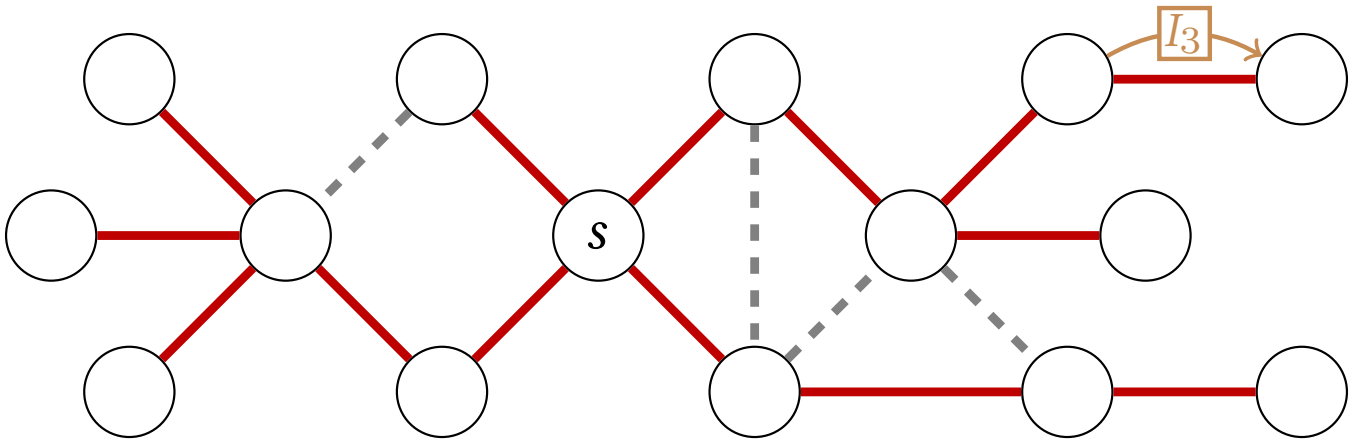
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

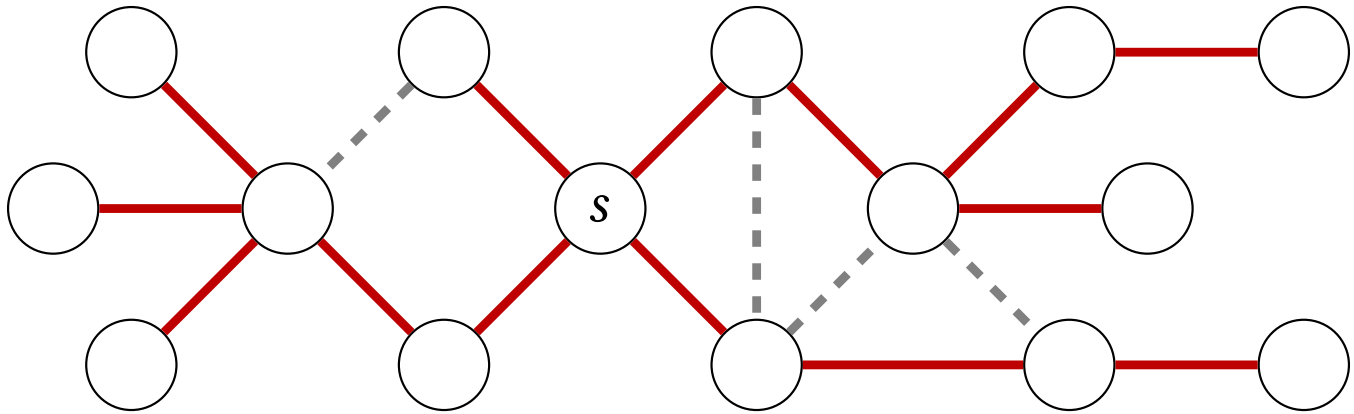
- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



Pipeline Algorithm

Algorithm:

- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children



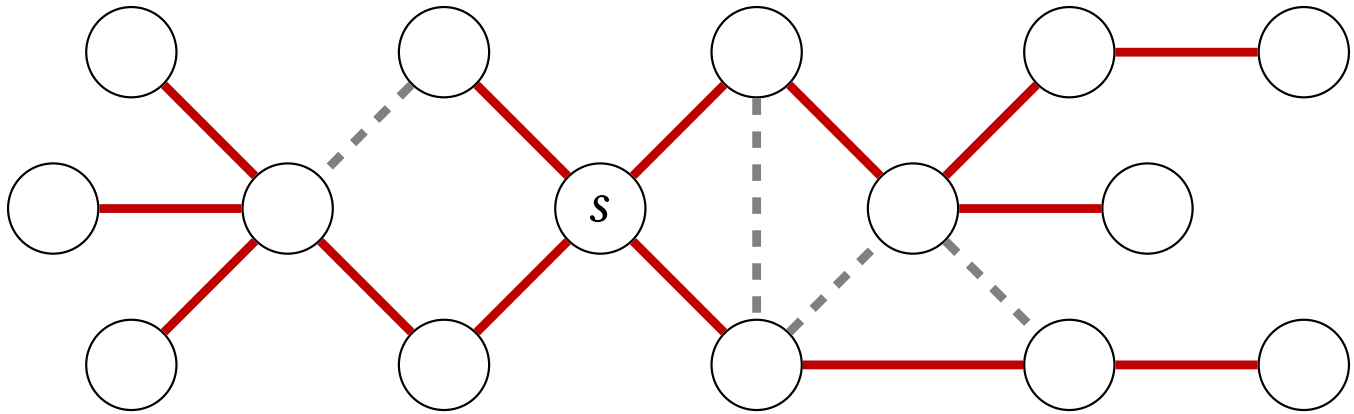
Induction claim

For every $1 \leq j \leq k$ and every node $v \neq s$, node v receives information I_j by round $\mathrm{dist}(s, v) + j$ at the latest.

Pipeline Algorithm

Algorithm:

- In round $j \leq k$, s sends information I_j to all neighbors
- Whenever some node $v \neq s$ receives information I for the first time: v sends I to all children

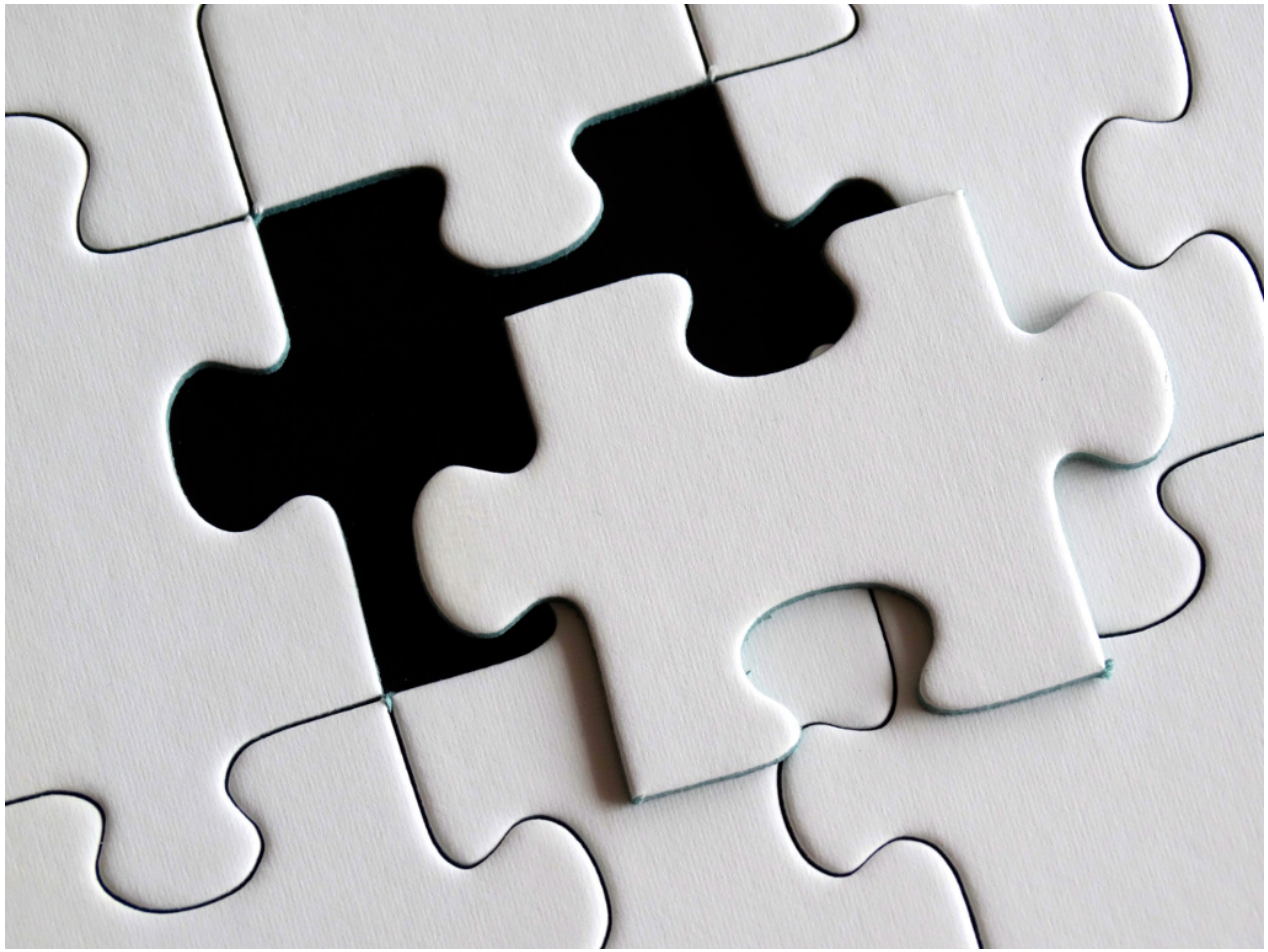


Induction claim

For every $1 \leq j \leq k$ and every node $v \neq s$, node v receives information I_j by round $\mathrm{dist}(s, v) + j$ at the latest.

$\Rightarrow$ **Running time** $O(\mathrm{Ecc}(s) + k) = O(D + k)$

Aggregation



Convergecast

Goal: s wants to ensure that the broadcast/breadth-first search is complete

Convergecast

Goal: s wants to ensure that the broadcast/breadth-first search is complete

Idea: Send an ACK (acknowledgment) message from every node to s via upcast

Convergecast

Goal: s wants to ensure that the broadcast/breadth-first search is complete

Idea: Send an ACK (acknowledgment) message from every node to s via upcast

Naive algorithm:

- Perform breadth-first search with spanning-tree construction
- Every node sends an ACK message to s via upcast in the tree

Convergecast

Goal: s wants to ensure that the broadcast/breadth-first search is complete

Idea: Send an ACK (acknowledgment) message from every node to s via upcast

Naive algorithm:

- Perform breadth-first search with spanning-tree construction
- Every node sends an ACK message to s via upcast in the tree

Problem: Congestion despite pipelining

Convergecast

Goal: s wants to ensure that the broadcast/breadth-first search is complete

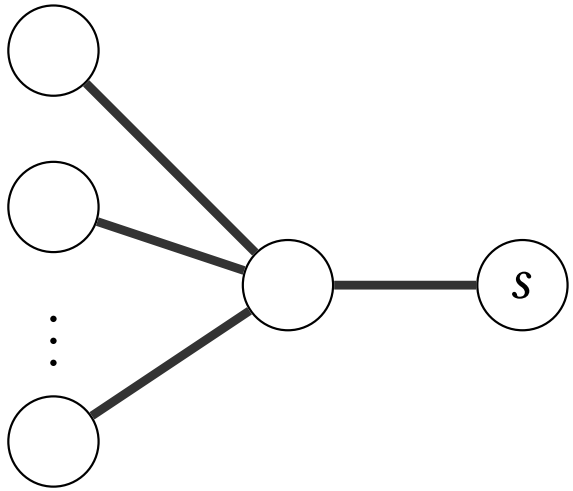
Idea: Send an ACK (acknowledgment) message from every node to s via upcast

Naive algorithm:

- Perform breadth-first search with spanning-tree construction
- Every node sends an ACK message to s via upcast in the tree

Problem: Congestion despite pipelining

$\Theta(n)$ rounds in the worst case!



ACK Message Aggregation

Algorithm:

- Perform breadth-first search with spanning-tree construction

ACK Message Aggregation

Algorithm:

- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent

ACK Message Aggregation

Algorithm:

- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no “child” message

ACK Message Aggregation

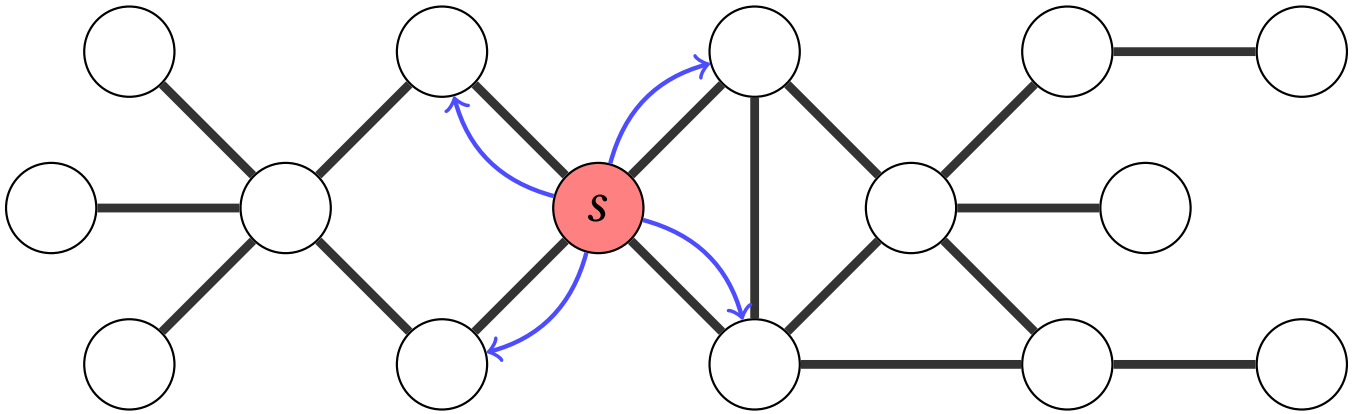
Algorithm:

- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children

ACK Message Aggregation

Algorithm:

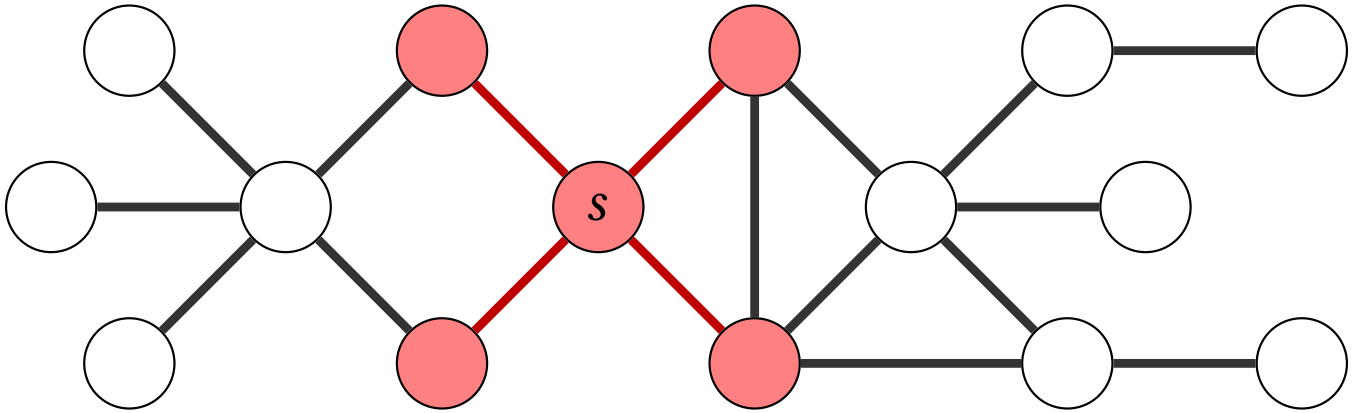
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

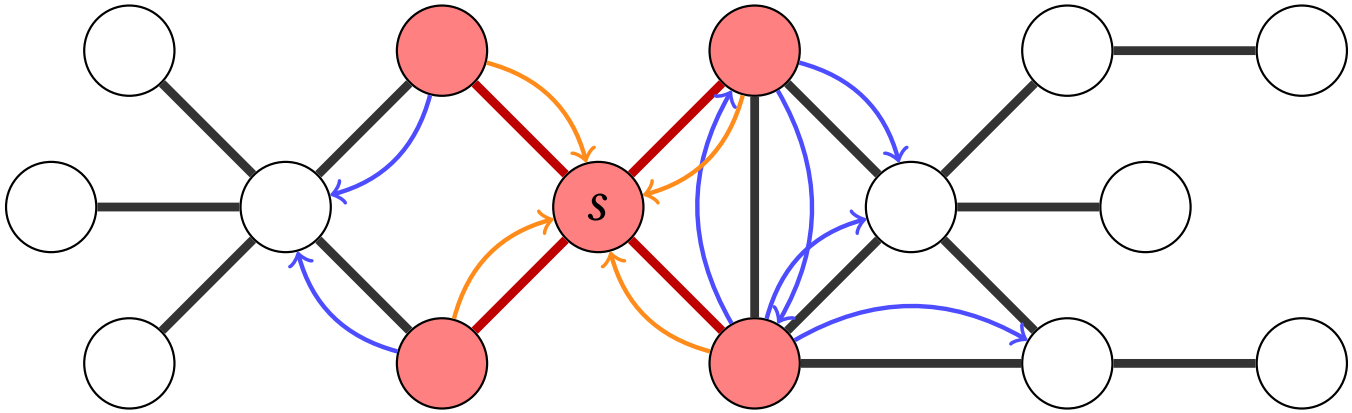
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

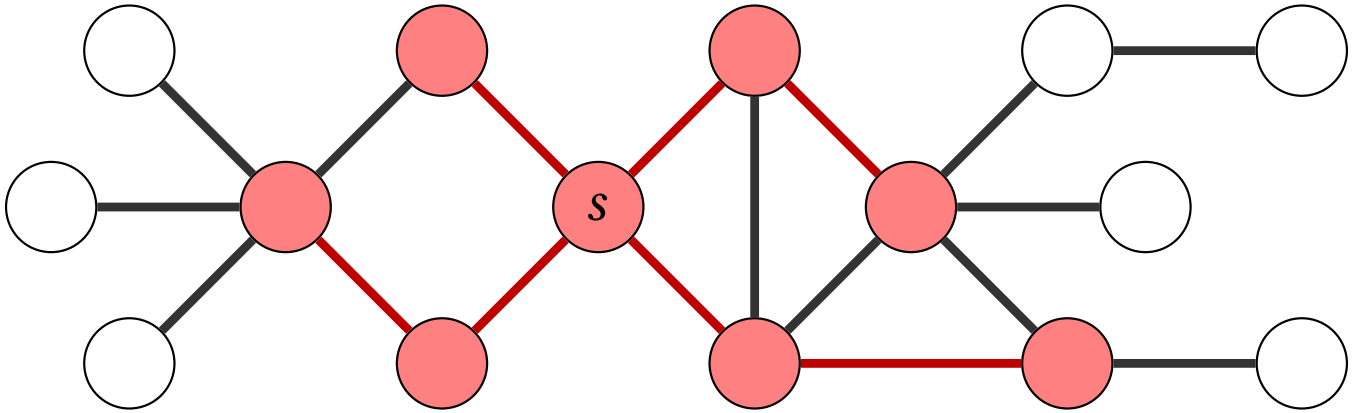
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

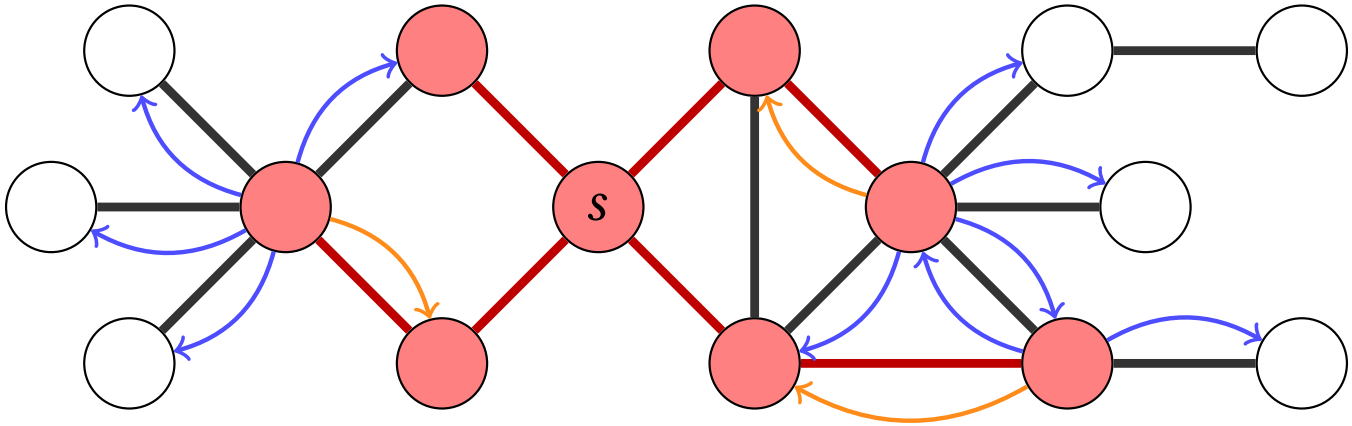
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

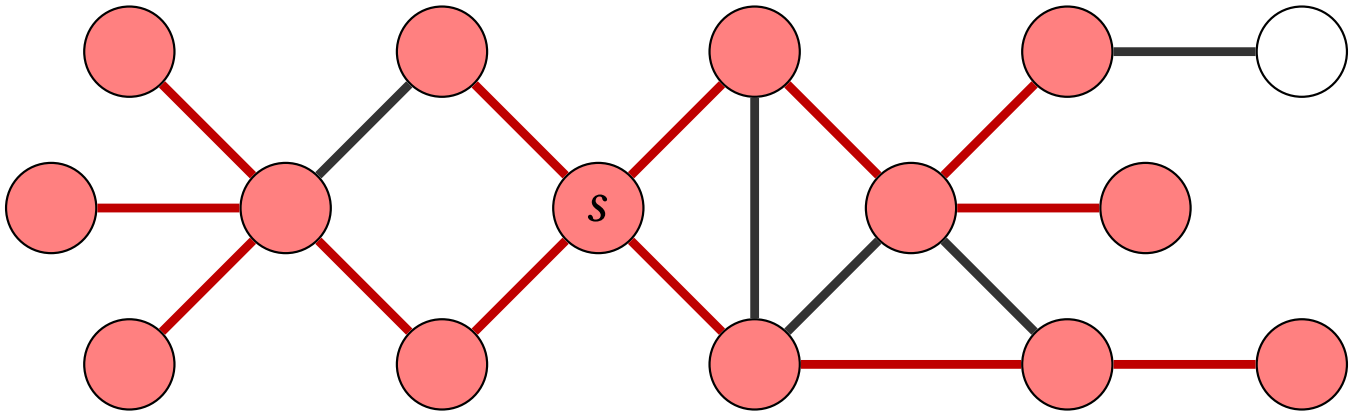
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

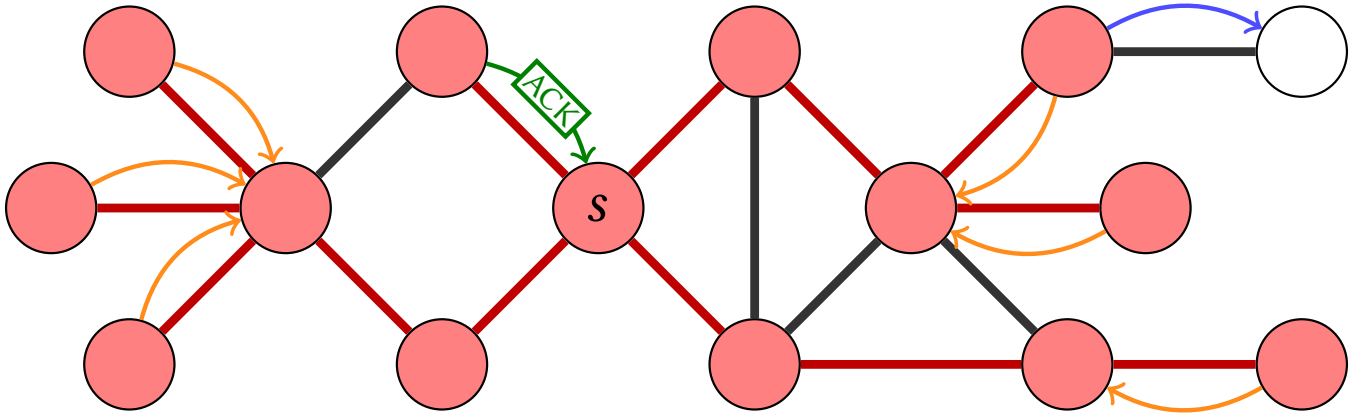
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

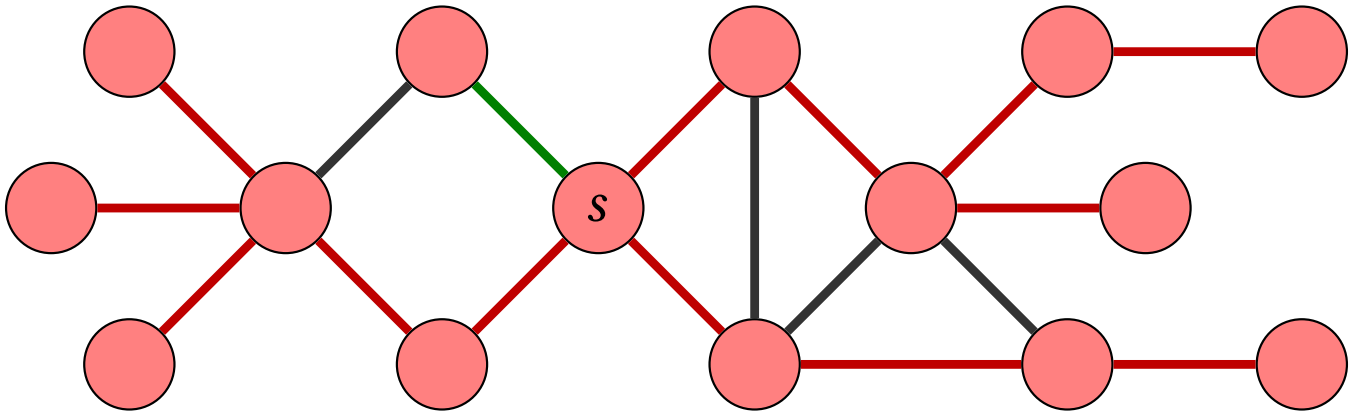
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

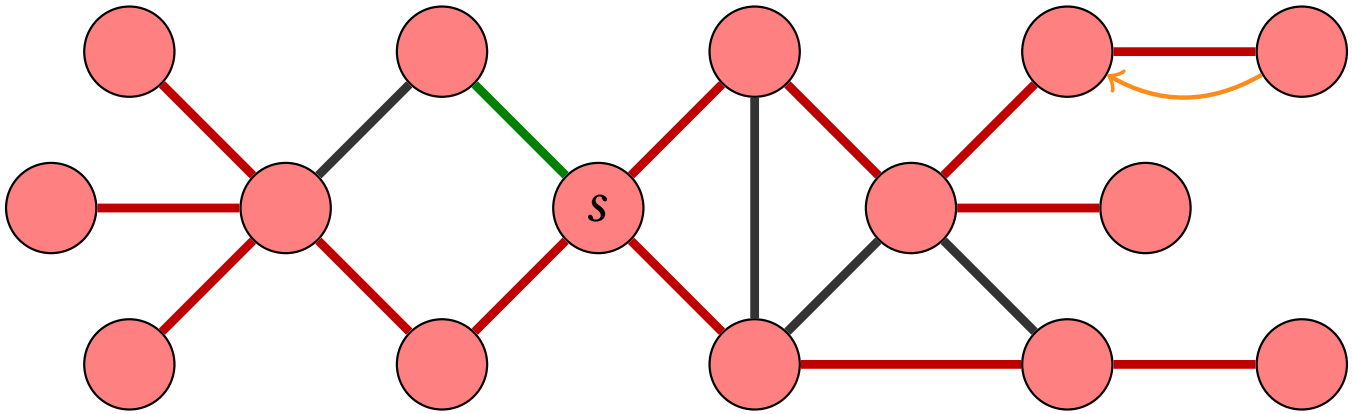
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

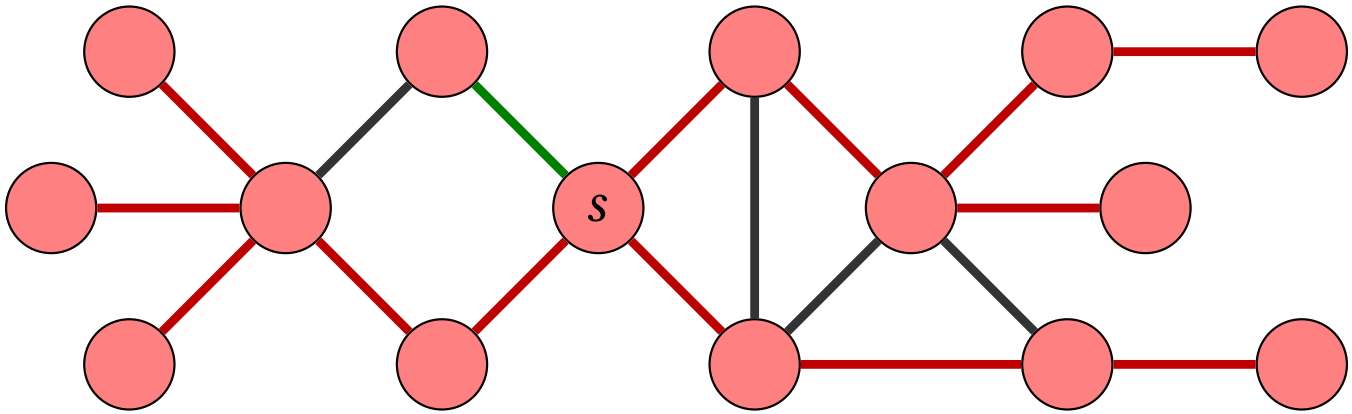
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

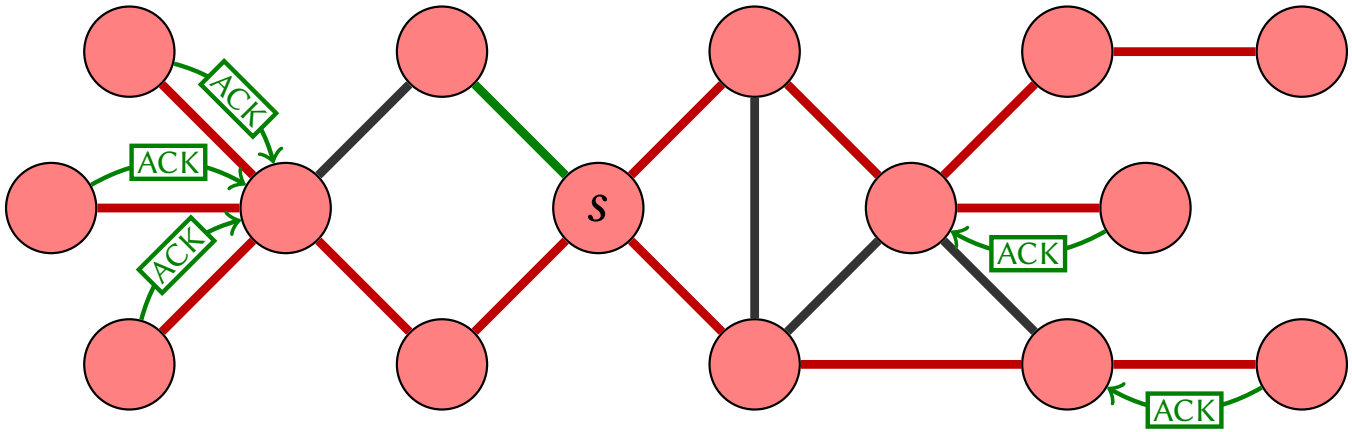
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

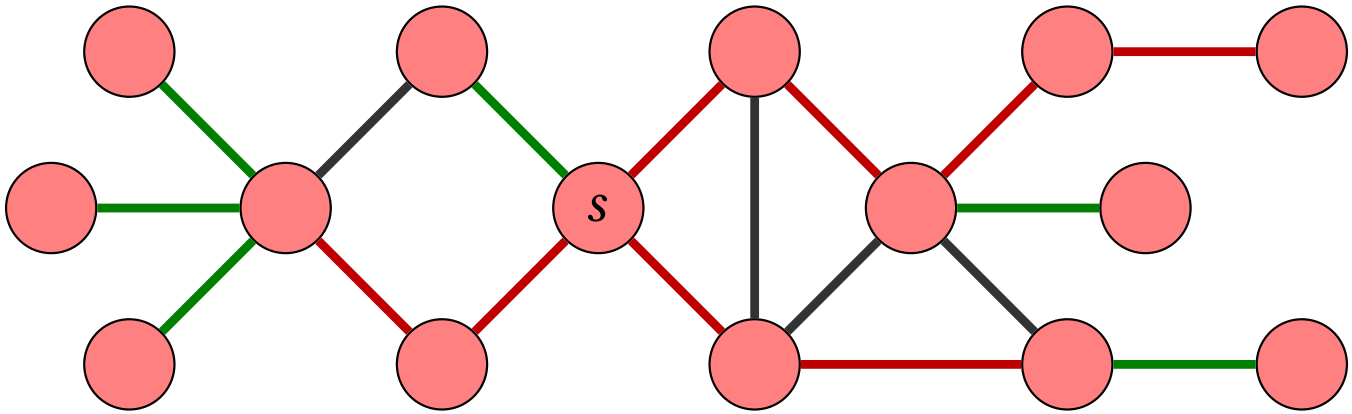
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

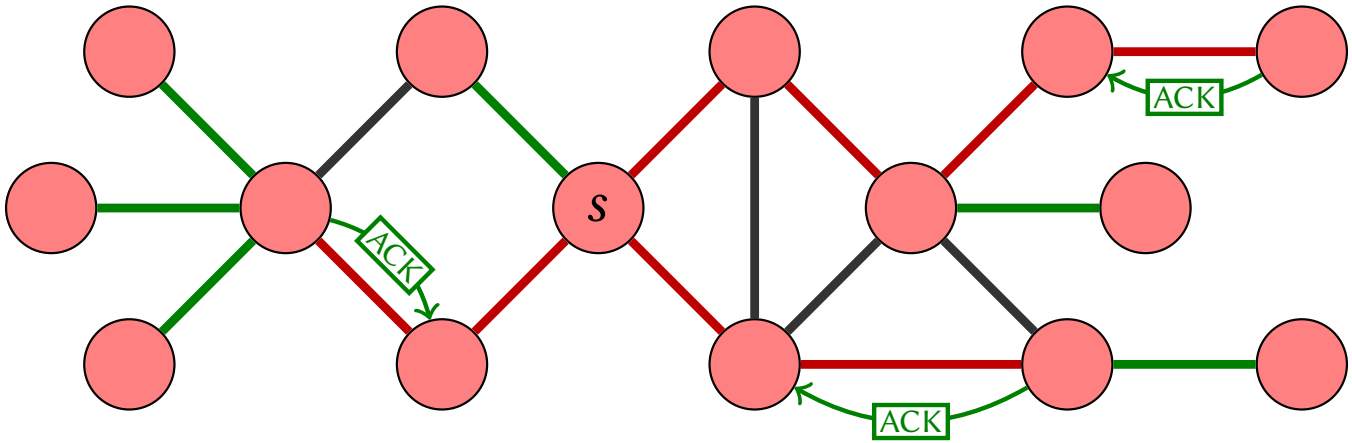
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

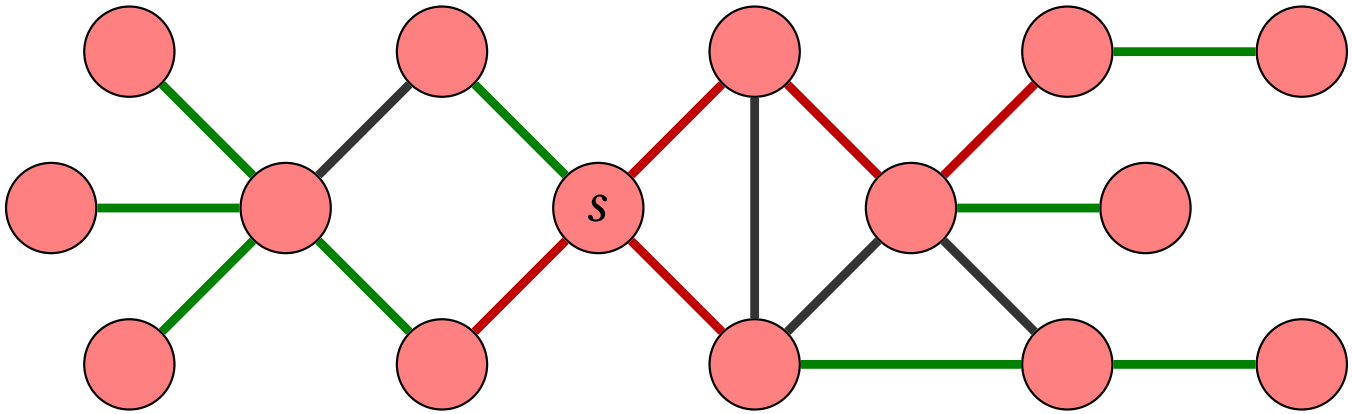
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

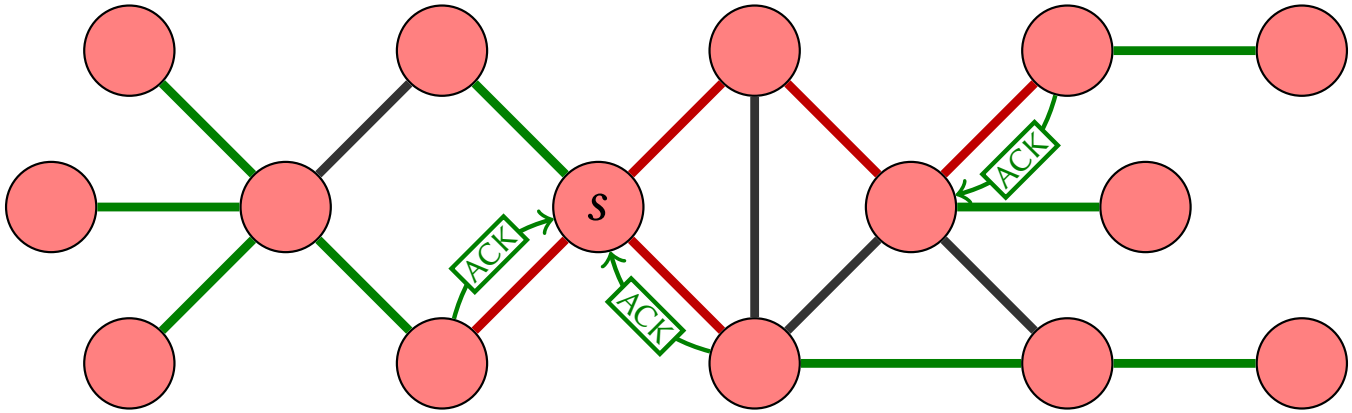
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

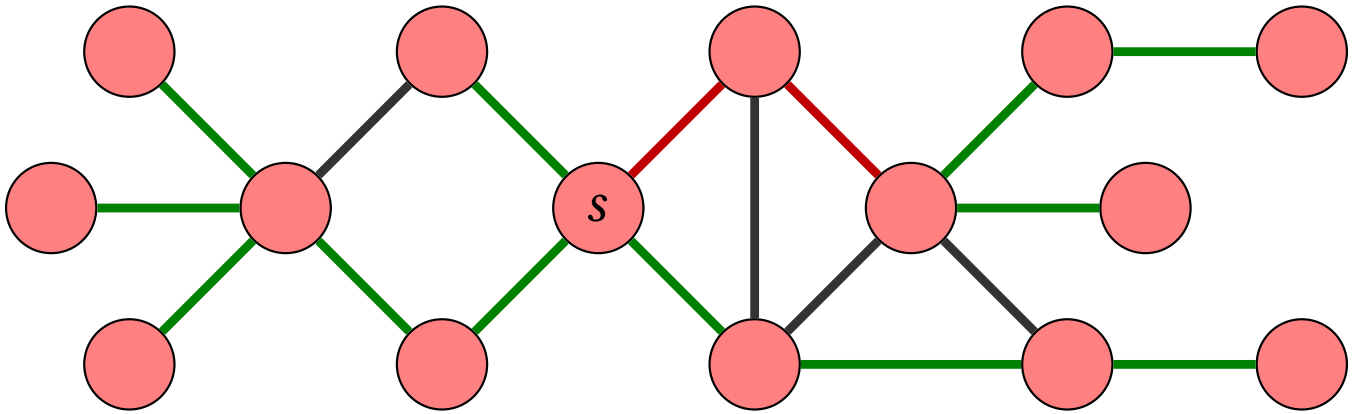
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

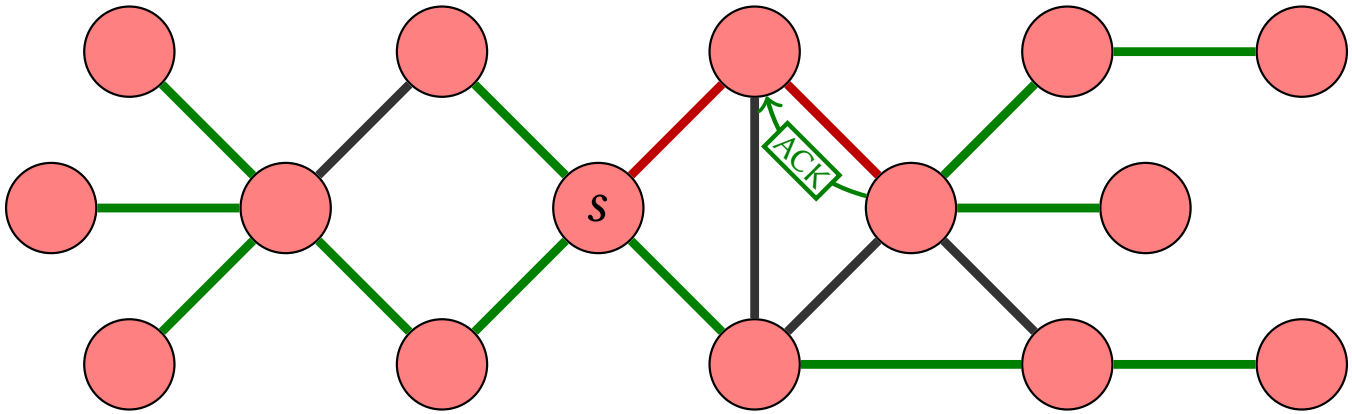
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

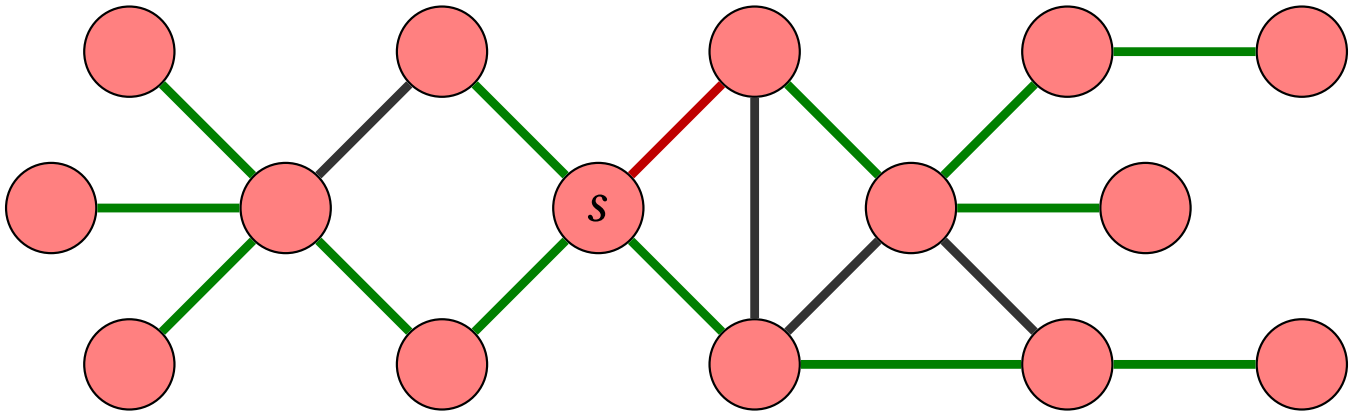
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

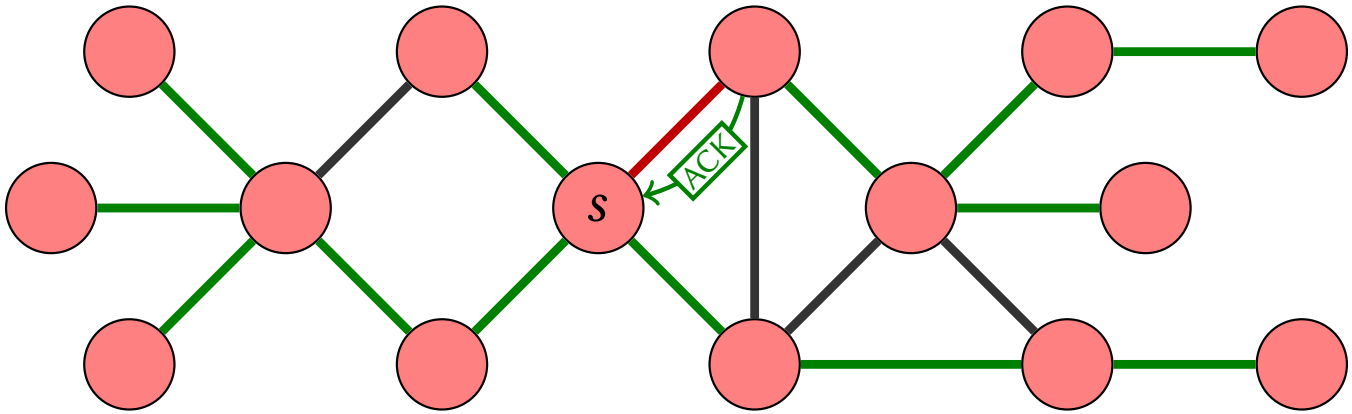
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

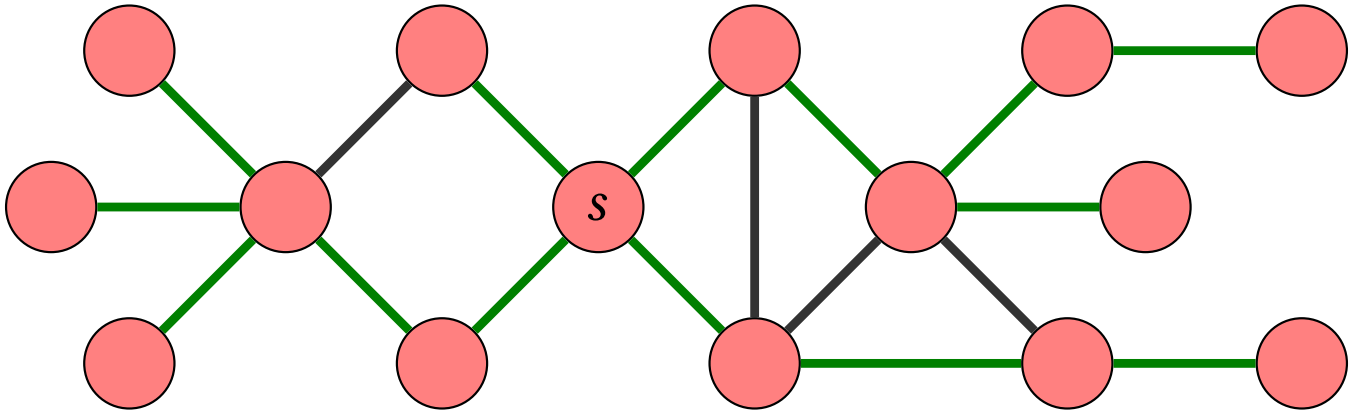
- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



ACK Message Aggregation

Algorithm:

- Perform breadth-first search with spanning-tree construction
- As soon as a node has received ACKs from all children: send an ACK to the parent
- Leaves in the tree recognize themselves as such if they receive no "child" message
- s waits until it has received ACKs from all children



Aggregation Functions

- Functional view: a node's ACK bit is the conjunction of the ACK bits of its children

Aggregation Functions

- Functional view: a node's ACK bit is the conjunction of the ACK bits of its children
- The tree structure enables bottom-up computation and waiting for the children's function values

Aggregation Functions

- Functional view: a node's ACK bit is the conjunction of the ACK bits of its children
- The tree structure enables bottom-up computation and waiting for the children's function values
- Other aggregation functions: sum, minimum, maximum, etc.

Queuing



Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent

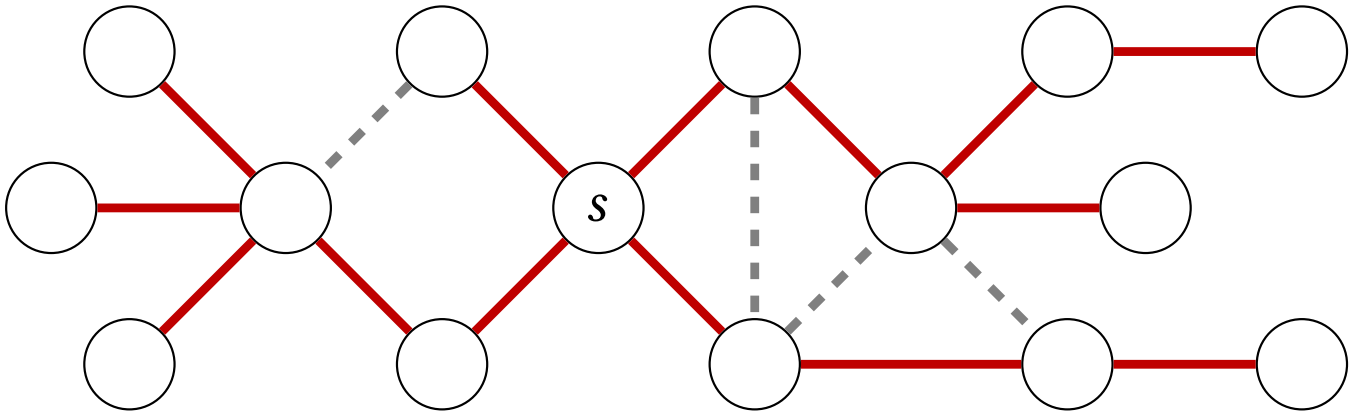
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



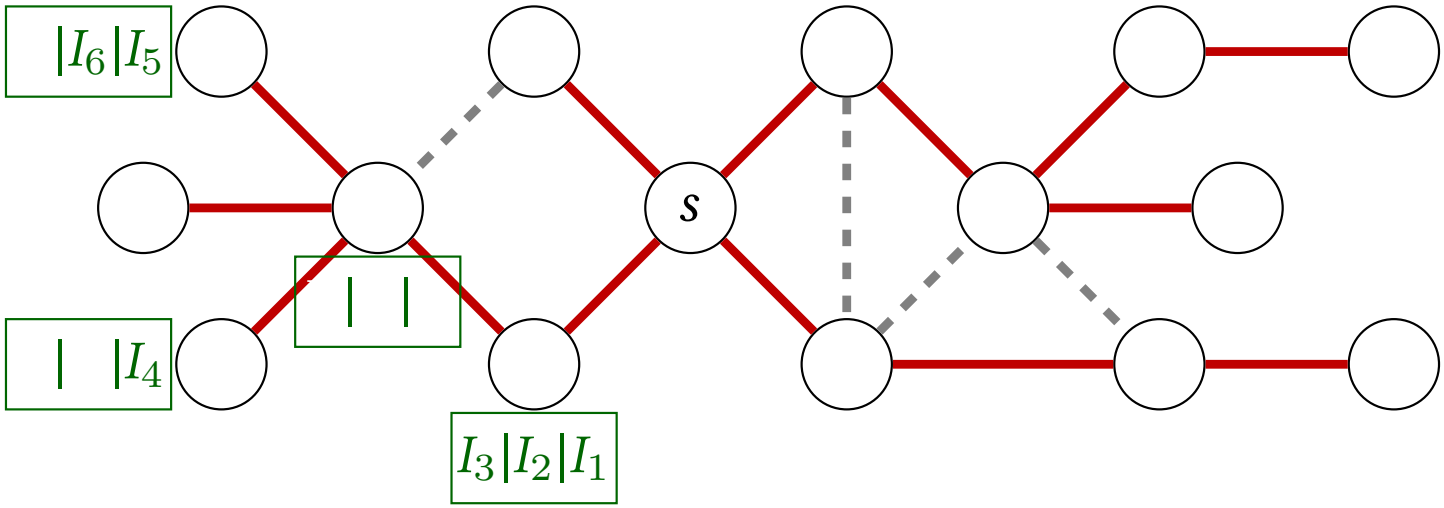
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



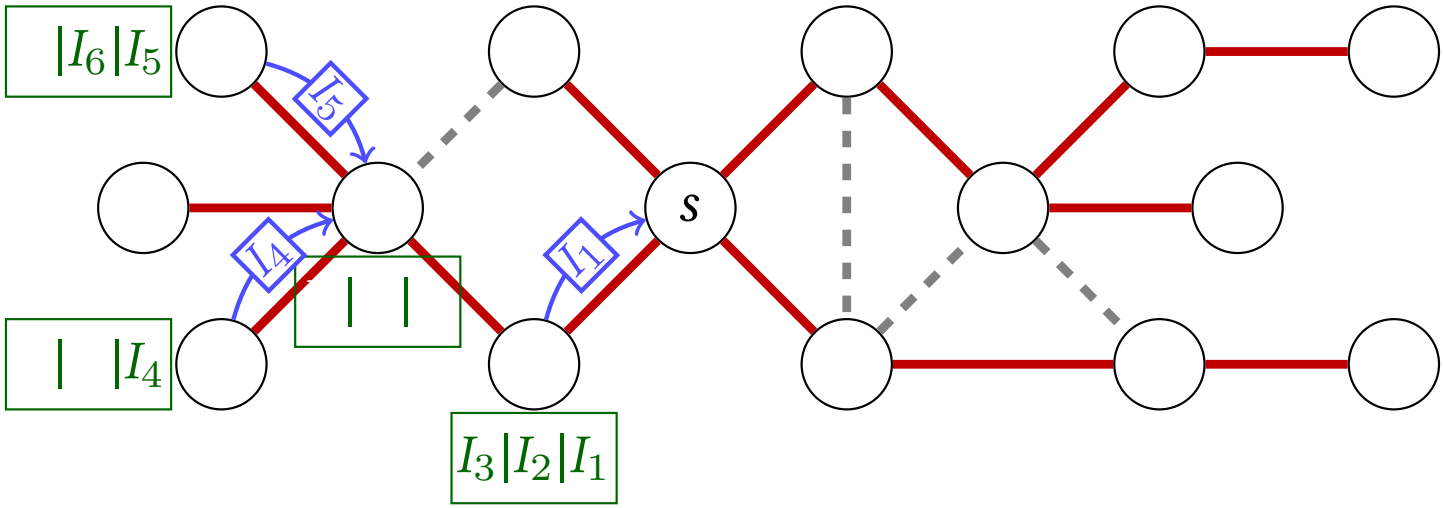
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



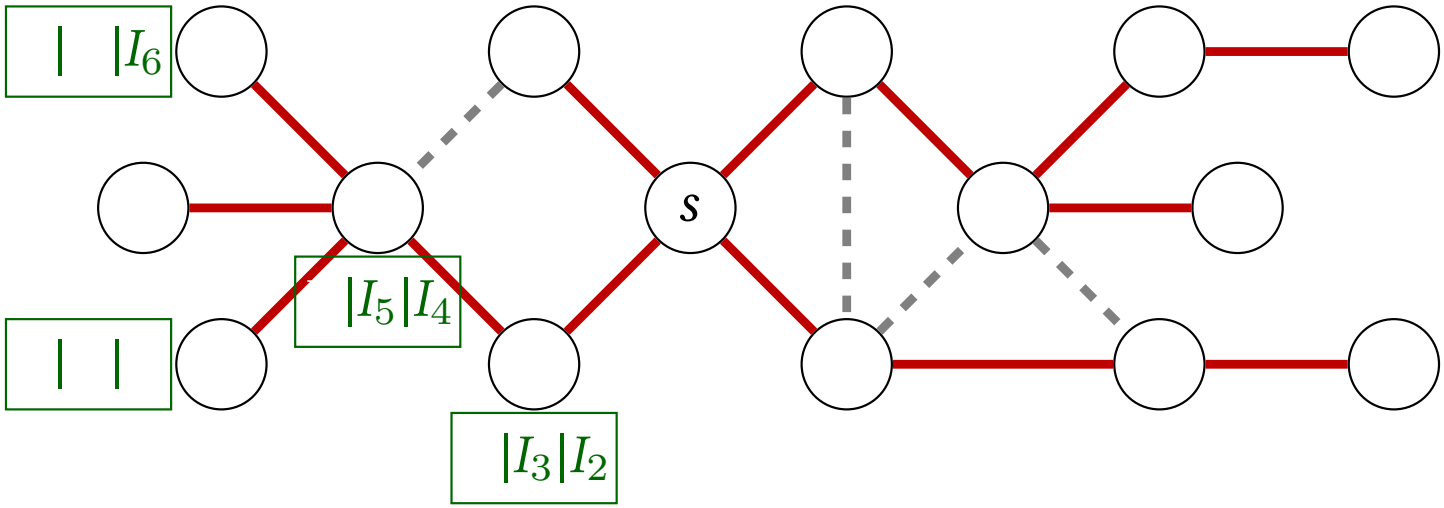
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



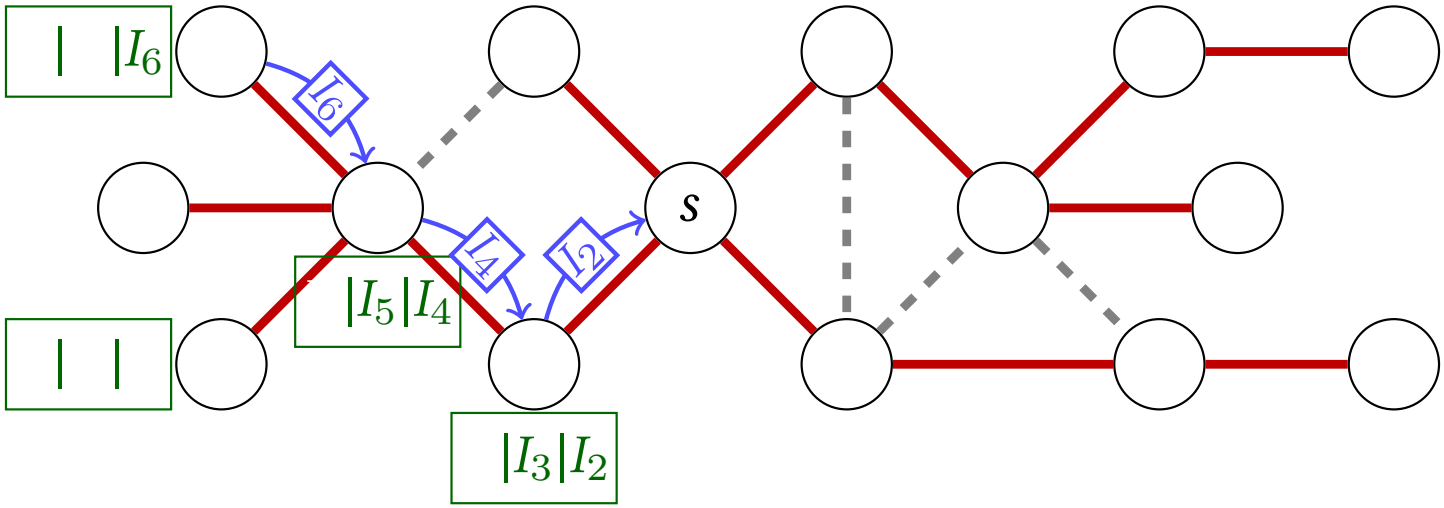
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



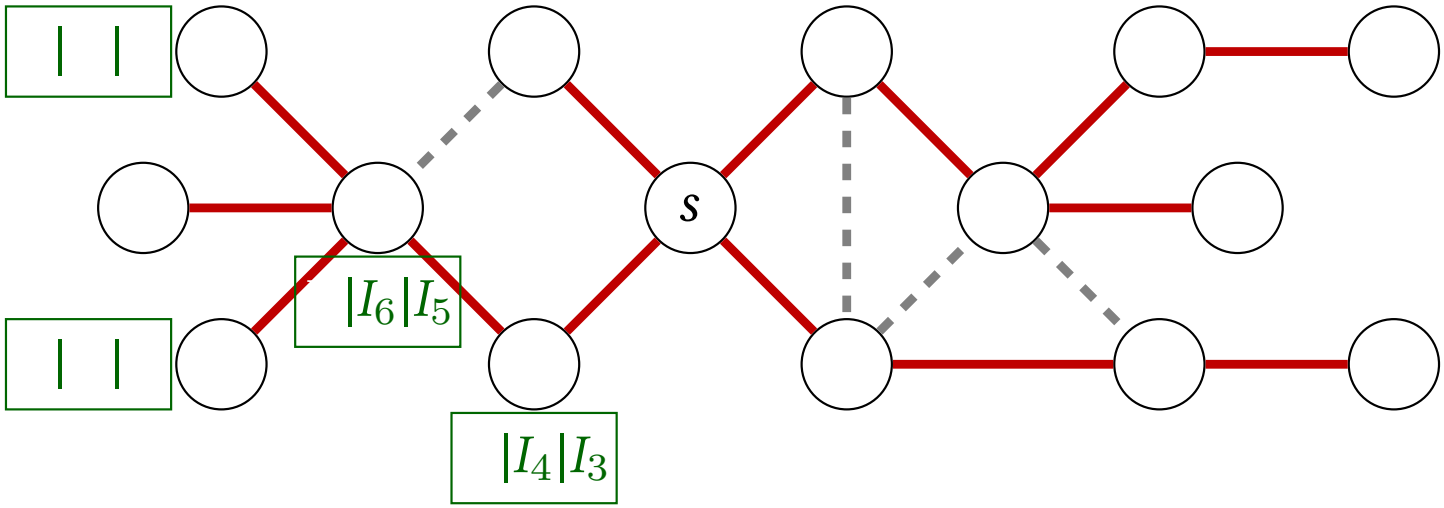
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



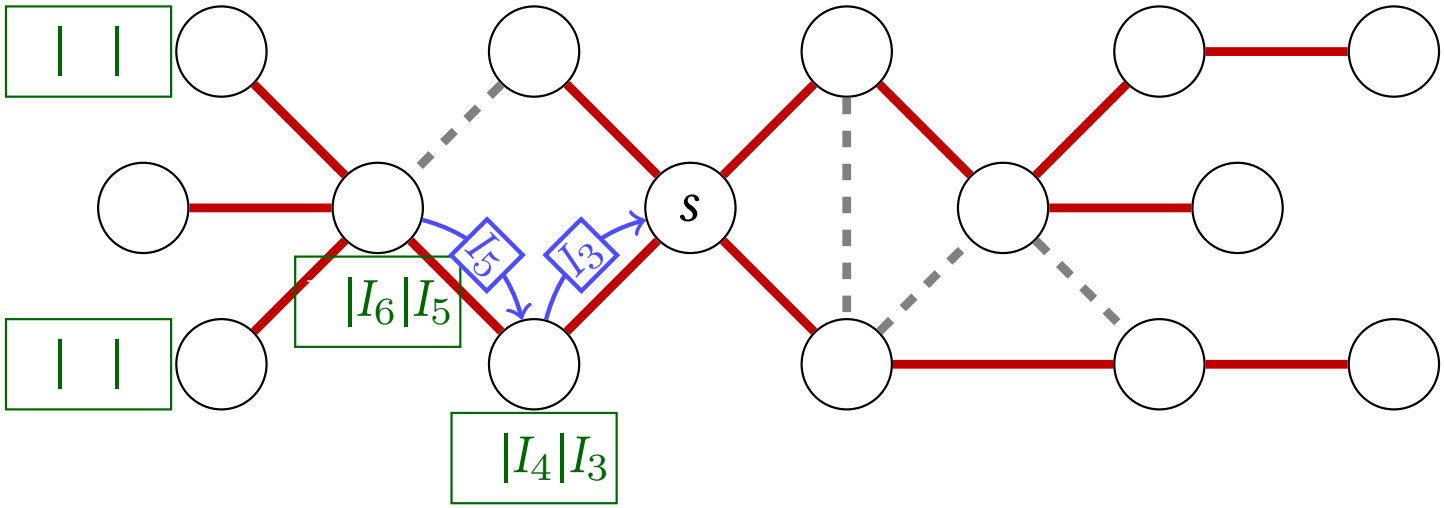
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



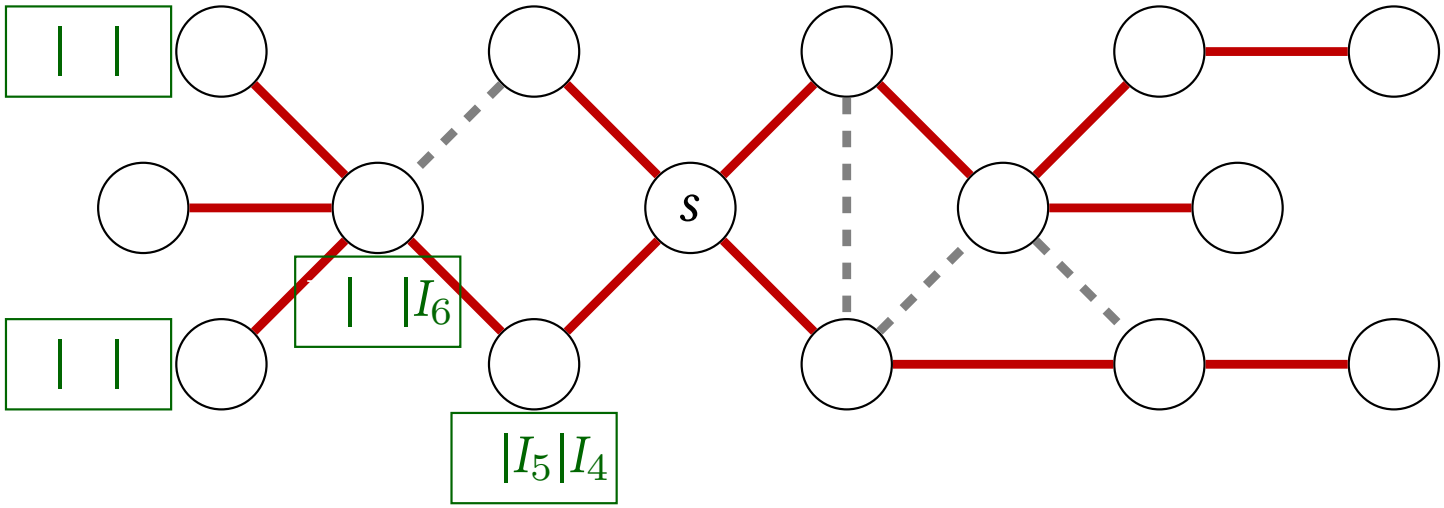
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



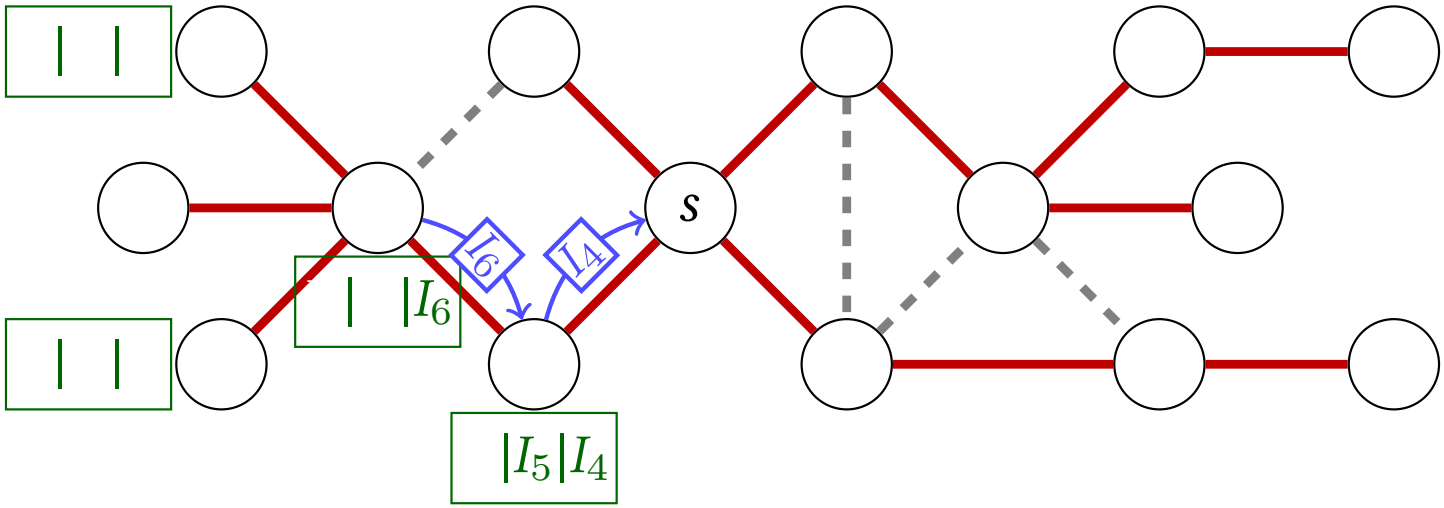
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



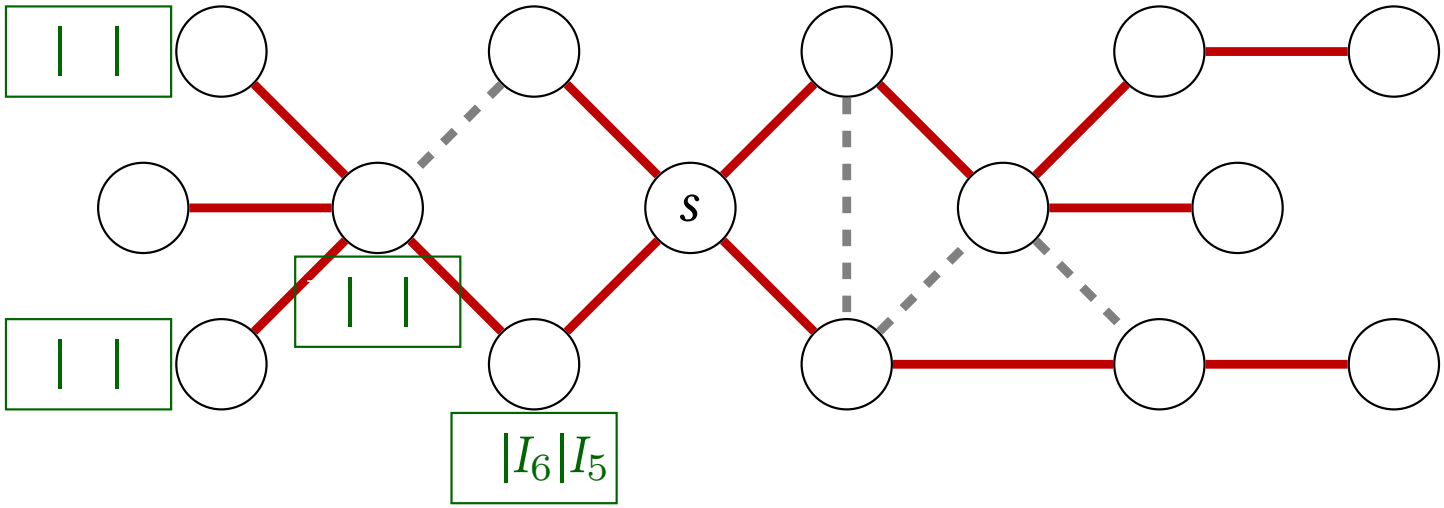
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



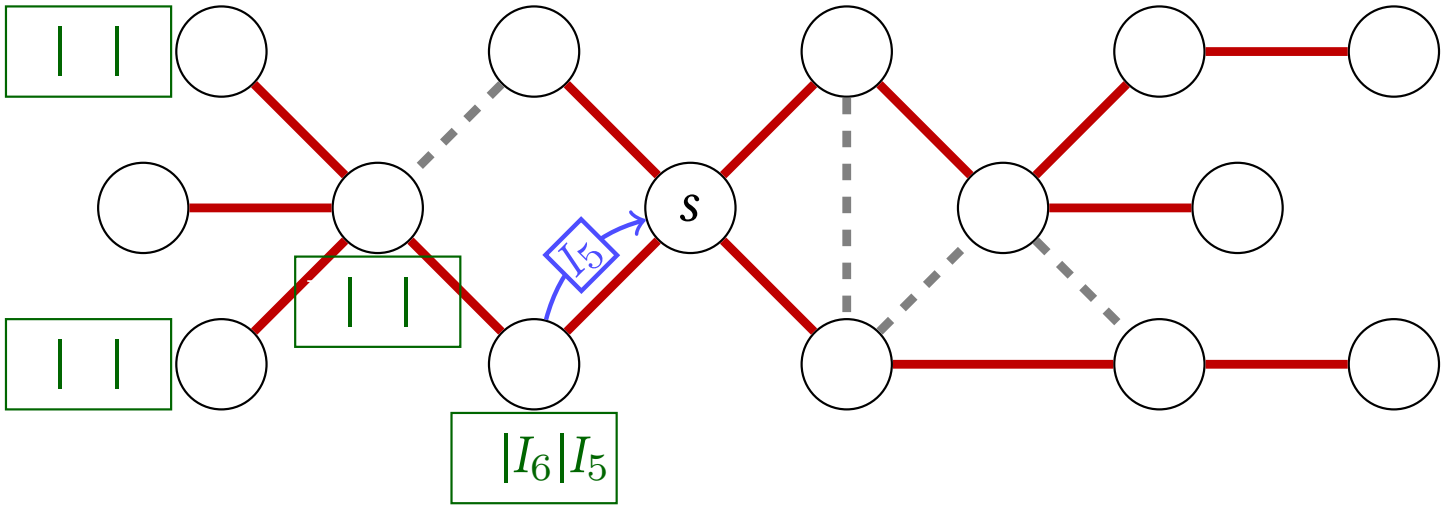
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



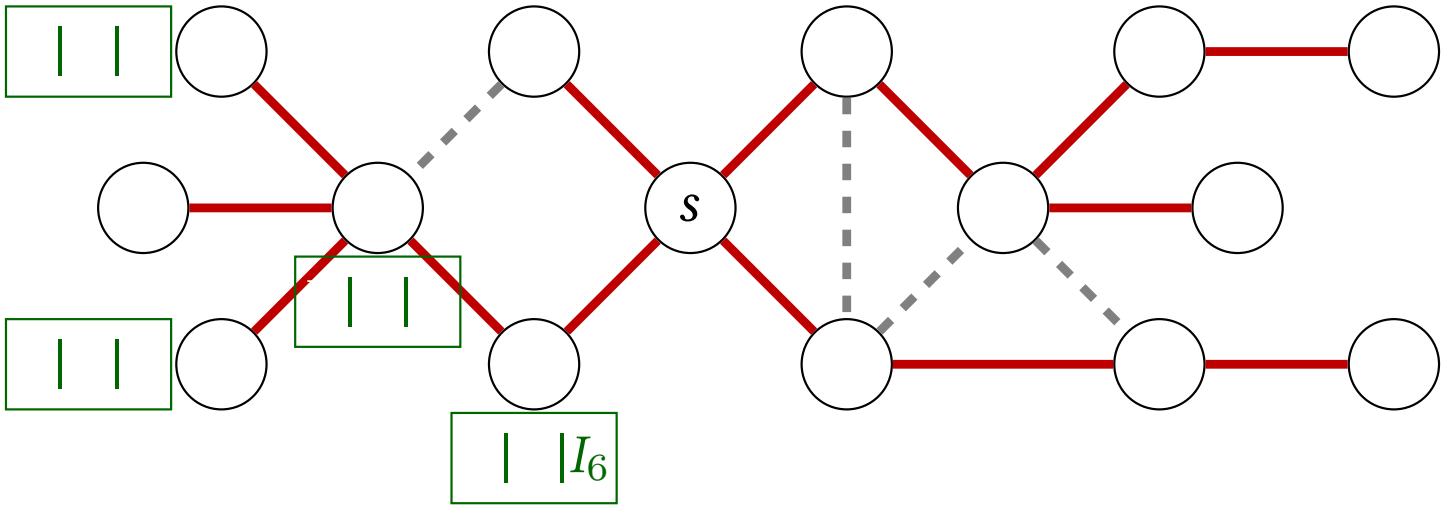
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



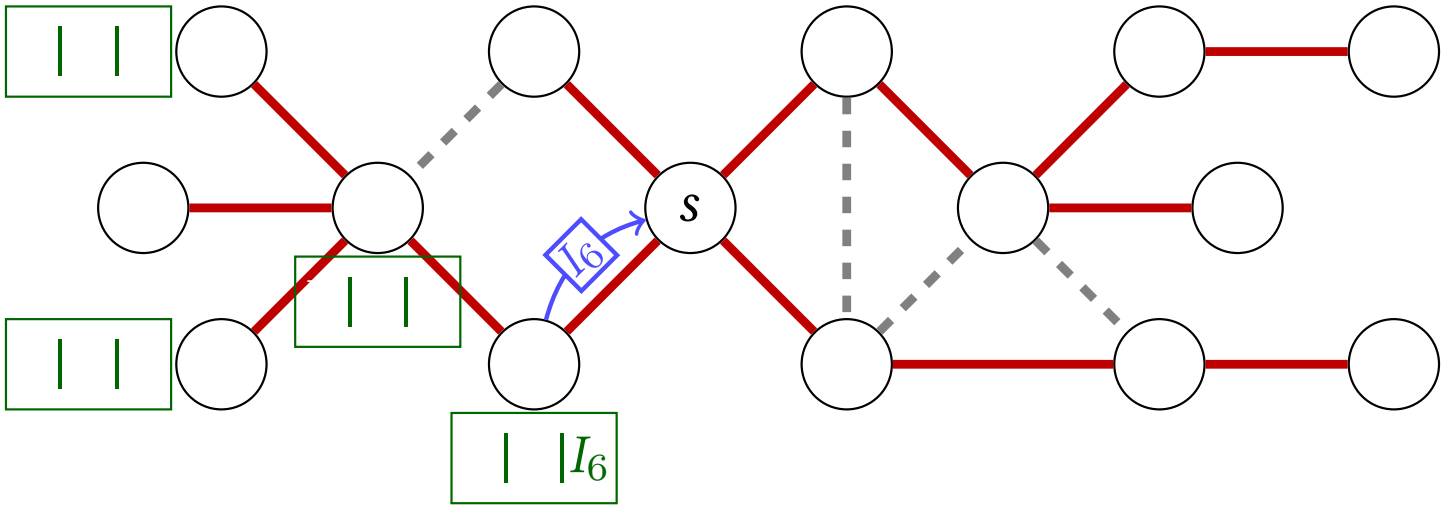
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



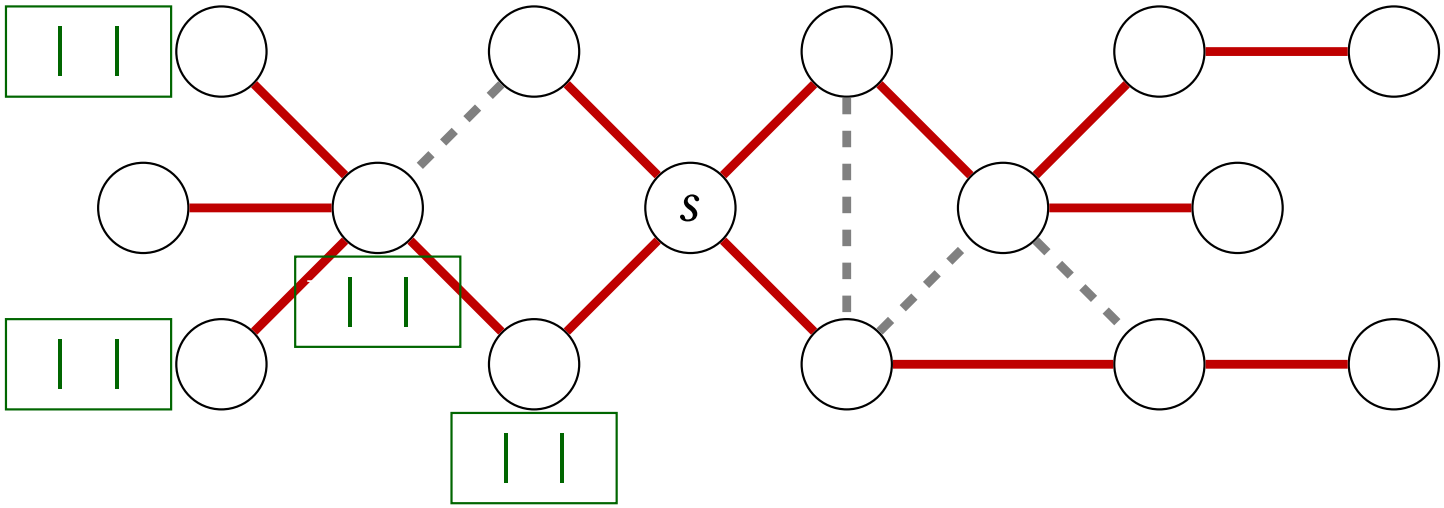
Multiple Upcast

Assumption: A BFS tree rooted at s has already been computed

Initial situation: The nodes of the network have k pieces of information $I_1, \ldots, I_k$ (each of size $O(\log n)$)
Goal: Node s has information $I_1, \ldots, I_k$

Algorithm:

- Every node has a queue of information not yet forwarded
- In every round, every node sends one piece of information from its queue to its parent



Running Time Bound with Auxiliary Lemma

Lemma

For every $r \geq 1$ *and every node* v *with* $\mathrm{dist}(s, v) \geq \mathrm{Ecc}(s) - r + 1$, *if* v *receives no information in round* r, *then it also receives no information in round* $r + 1$.

Running Time Bound with Auxiliary Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Theorem

The multiple-upcast algorithm requires $O(D+k)$ rounds.

Running Time Bound with Auxiliary Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Theorem

The multiple-upcast algorithm requires $O(D+k)$ rounds.

Proof:

- Starting in round $r = \mathrm{Ecc}(s) + 1$, we have $\mathrm{dist}(s,s) = 0 \geq \mathrm{Ecc}(s) - r + 1$; therefore, s must receive one piece of information in every round, or s receives no further information and the algorithm terminates.

Running Time Bound with Auxiliary Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Theorem

The multiple-upcast algorithm requires $O(D+k)$ rounds.

Proof:

- Starting in round $r = \mathrm{Ecc}(s) + 1$, we have $\mathrm{dist}(s,s) = 0 \geq \mathrm{Ecc}(s) - r + 1$; therefore, s must receive one piece of information in every round, or s receives no further information and the algorithm terminates.
- Each of the k pieces of information is received by every node, and therefore also by s, at most once.

Running Time Bound with Auxiliary Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Theorem

The multiple-upcast algorithm requires $O(D+k)$ rounds.

Proof:

- Starting in round $r = \mathrm{Ecc}(s) + 1$, we have $\mathrm{dist}(s,s) = 0 \geq \mathrm{Ecc}(s) - r + 1$; therefore, s must receive one piece of information in every round, or s receives no further information and the algorithm terminates.
- Each of the k pieces of information is received by every node, and therefore also by s, at most once.
- **Thus:** At most $\mathrm{Ecc}(s) + k = O(D+k)$ rounds.

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r-1) + 1$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r-1) + 1$
    - ★ By the induction hypothesis: u receives no information in round $(r-1)+1 = r$

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r-1) + 1$
    - ★ By the induction hypothesis: u receives no information in round $(r-1)+1 = r$
    - ★ Thus, queue of u is still empty in round r, and u sends no information to v

Proof of the Lemma

Lemma

For every $r \geq 1$ and every node v with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$, if v receives no information in round r, then it also receives no information in round $r+1$.

Proof:

- **Base case** ($r = 1$):
  - Let v be a node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1 = \mathrm{Ecc}(s)$
  - Then v is a leaf in the BFS tree and receives no information
- **Inductive step** ($r \geq 2$):
  - Let v be a non-leaf node with $\mathrm{dist}(s,v) \geq \mathrm{Ecc}(s) - r + 1$
  - Let u be a child of v in the BFS tree
    - ★ Since v receives no information in round r, u sent no information to v in round $r-1$
    - ★ Therefore, u had an empty queue in round $r-1$ and received no information
    - ★ We have: $\mathrm{dist}(s,u) = \mathrm{dist}(s,v) + 1 \geq \mathrm{Ecc}(s) - r + 1 + 1 = \mathrm{Ecc}(s) - (r-1) + 1$
    - ★ By the induction hypothesis: u receives no information in round $(r-1)+1 = r$
    - ★ Thus, queue of u is still empty in round r, and u sends no information to v
  - **Thus:** v receives no information in round $r+1$

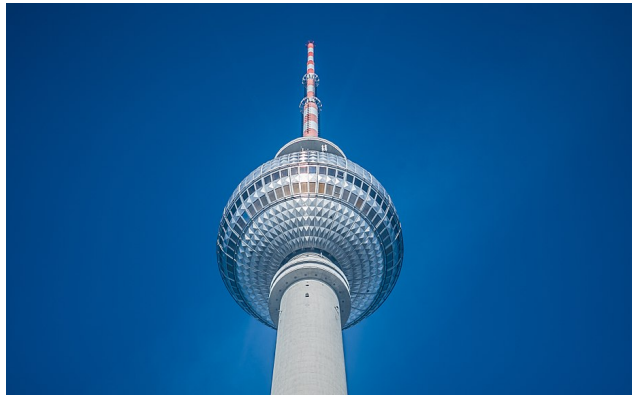
Summary

Summary



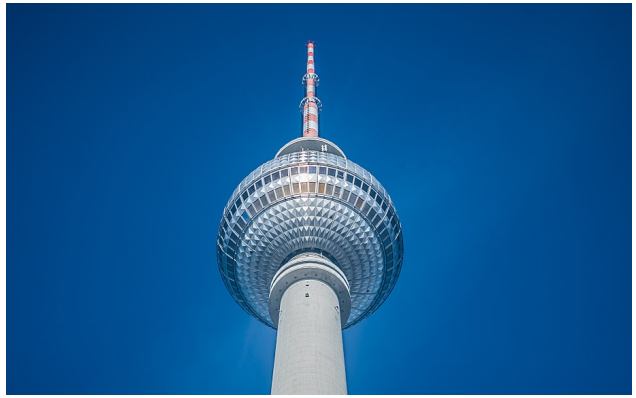
Summary





Summary

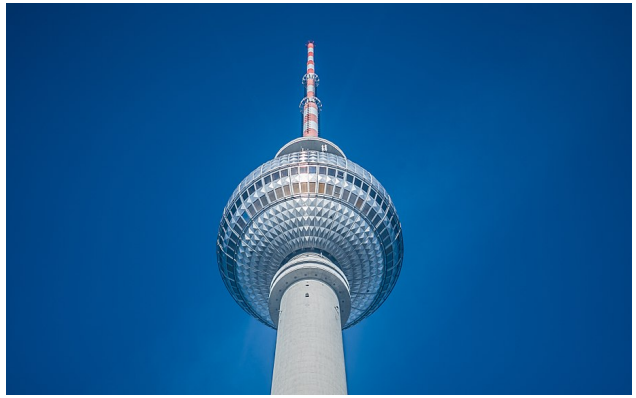






Summary



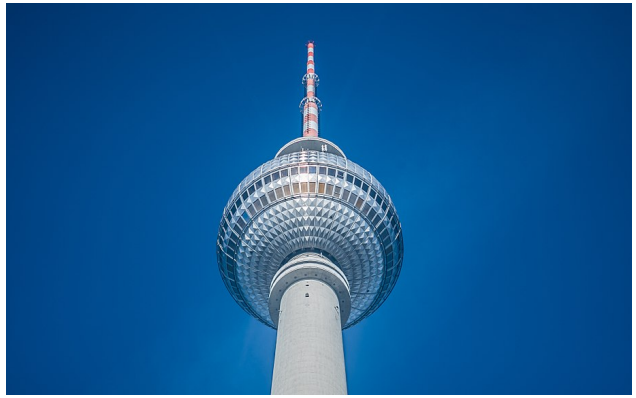






Summary





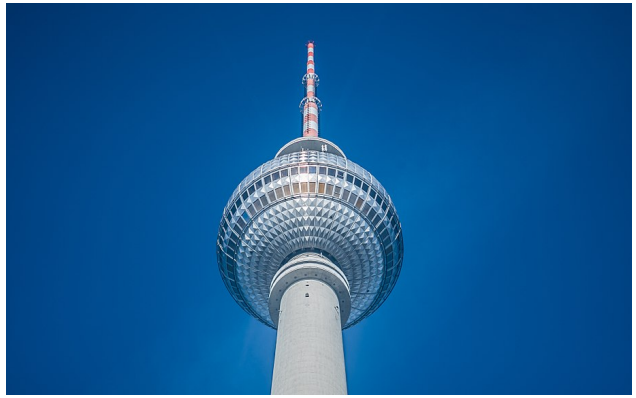






Summary













References

The content of this lecture unit is partly based on a lecture by Danupon Nanongkai.

Literature:

- David Peleg (2000) *Distributed Computing*, Chapters 3 and 4, SIAM.