Chapter 3

Broadcast, Convergecast, and Spanning Trees

Distributed Systems

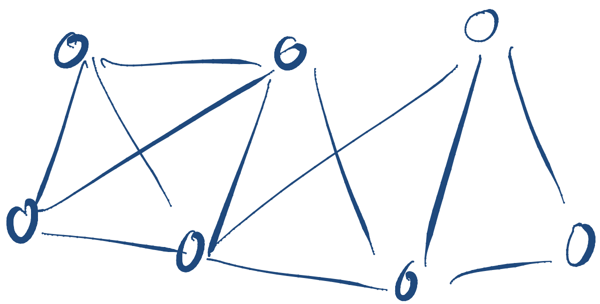
SS 2015

Fabian Kuhn

Message Passing in Arbitrary Topologies

Assumption for this chapter:

- Network: message passing system with arbitrary topology
- network topology is given by an undirected graph $G = (V, E)$



- Only overlap with "Network Algorithms" lecture
  – with the lecture this morning...

Asynchronous Message Passing

In this chapter: No failures, but asynchrony

Asynchronous message passing:

- messages are always delivered in finite time
  - cf.: finite time → admissible schedule
- message delays are completely unpredictable
- algorithms are **event-based**:

upon receiving message from neighbor ..., do

some local computation steps

send message(s) to neighbor(s) ...

Broadcast

- Simple, basic communication problem

Problem Description:

- A source node s needs to broadcast a message M to all nodes of the system (network)
- Each node has a unique ID
- Initially, each node knows the IDs of its neighbors
  - or it can count / distinguish its neighbors by individual communication ports to the pairwise communication links

Flooding

- One of the simplest distributed (network) algorithms

Basic idea:

- When receiving M for the first time, forward to all neighbors

Algorithm:

- Source node s:
  initially do
  send M to all neighbors

- All other nodes u:
  upon receiving M **from some neighbor** v
  if M has not been received before **then**
  send M to all neighbors except v

Flooding in Synchronous Systems

Synchronous systems:

- time divided into synchronous rounds, msg. delay $= 1$ round
- time complexity: number of rounds

Progress in flooding algorithm:

Flooding in Synchronous Systems

Synchronous systems:

- time divided into synchronous rounds, msg. delay $=$ 1 round
- time complexity: number of rounds

Progress in flooding algorithm:

- after 1 round:
  - all neighbors of s know M
  - nodes at distance ≥ 2 from s do not know M
- after 2 rounds:
  - exactly nodes at distance ≤ 2 from s know M
- ...
- after r rounds:
  - exactly nodes at distance $\leq r$ from s know M

total time:
max. dist. of any node from s

Flooding in Synchronous Systems

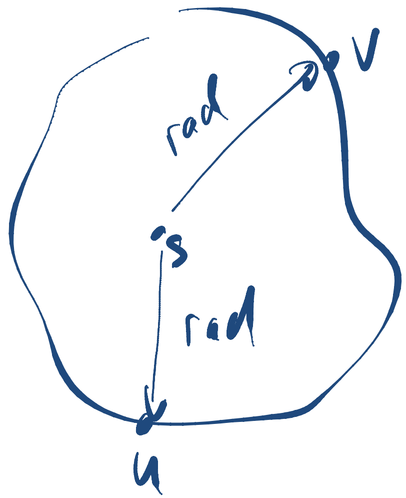
Radius: (Graph $G = (V, E)$)

- Given a node $s \in V$, radius of s in G:

$$rad(G, s) := \max_{v \in V} dist_G(s, v)$$

- radius of G:

$$rad(G) := \min_{s \in V} rad(G, s)$$



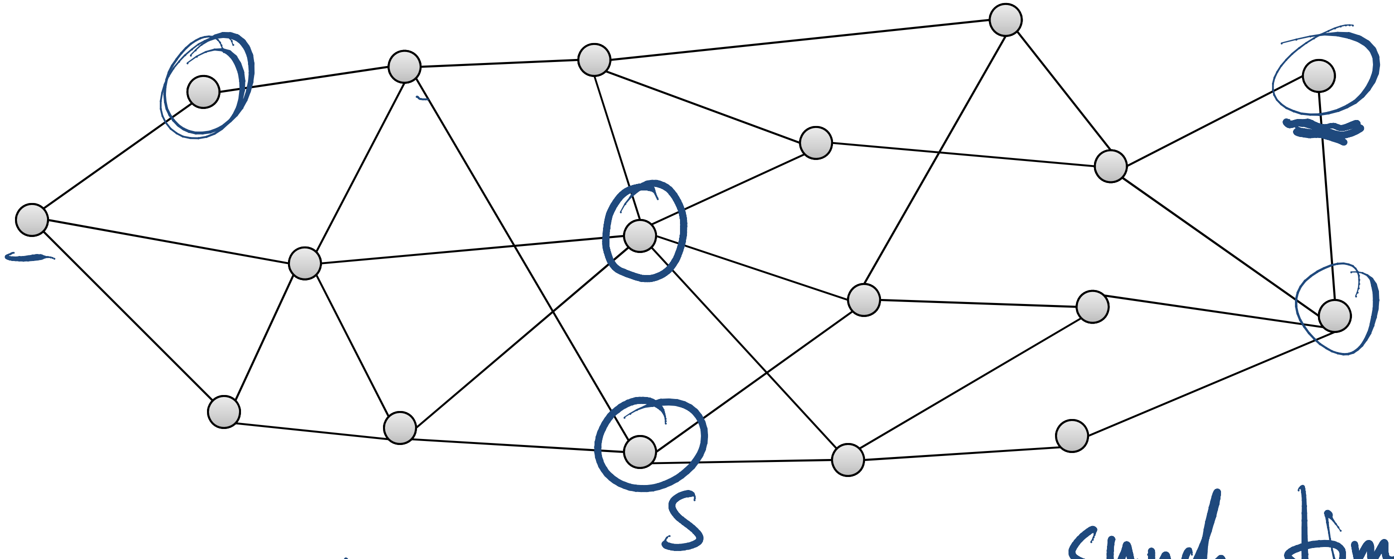
Diameter of G:

$$diam(G) := \max_{u,v \in V} dist_G(u, v) = \max_{s \in V} rad(G, s)$$

Time complexity of **flooding** in synchronous systems: $\boldsymbol{rad(G, s)}$

$$\frac{diam(G)}{2} \leq rad(G) \leq rad(G, s) \leq diam(G)$$

Radius and Diameter



rad(G,s) = 4

rad(G) = $\min_v$ rad(G,v) = 3

diam(G) = 6 or 5

synch. time compl. ≤ D

Asynchronous Time Complexity

- Time complexity of flooding in asynchronous systems?
- How is time complexity in asynchronous systems defined?

Assumptions:

- Message delays, time for local computations are arbitrary
  - Algorithms cannot use any timing assumptions!
- **For analysis:**
  - message delays ≤ 1 time unit
  - local computations take 0 time

Determine asynchronous time complexity:

1. determine running time of a given execution
2. **asynch. time complexity $=$ max. running time of any exec.**

Asynchronous Time Complexity

Running time of an execution:

- assign times to send and receive events such that
  - order of all events remains unchanged
  - local computations take 0 time
  - message delays are at most 1 time unit
  - first send event is at time 0
  - time of last event is maximized

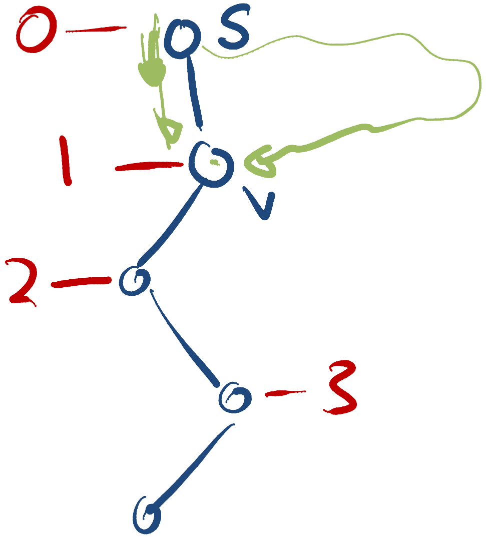
- essentially: normalize message delays such that the maximum delay is 1 time unit

Definition Asynchronous Time Complexity:

Total time of a worst-case execution in which local computations take time 0 and all message delays are at most 1 time unit.

Flooding in Asynchronous Systems

Theorem: The time complexity of flooding from a source s in an ansynchronous network G is $rad(G, s)$.



Message Complexity of Flooding

Message Complexity: Total number of messages sent

- total number of messages, over all nodes

What is the message complexity of flooding?

≤ 2 msg. per edge

Theorem: The message complexity of flooding is $O(|E|)$.

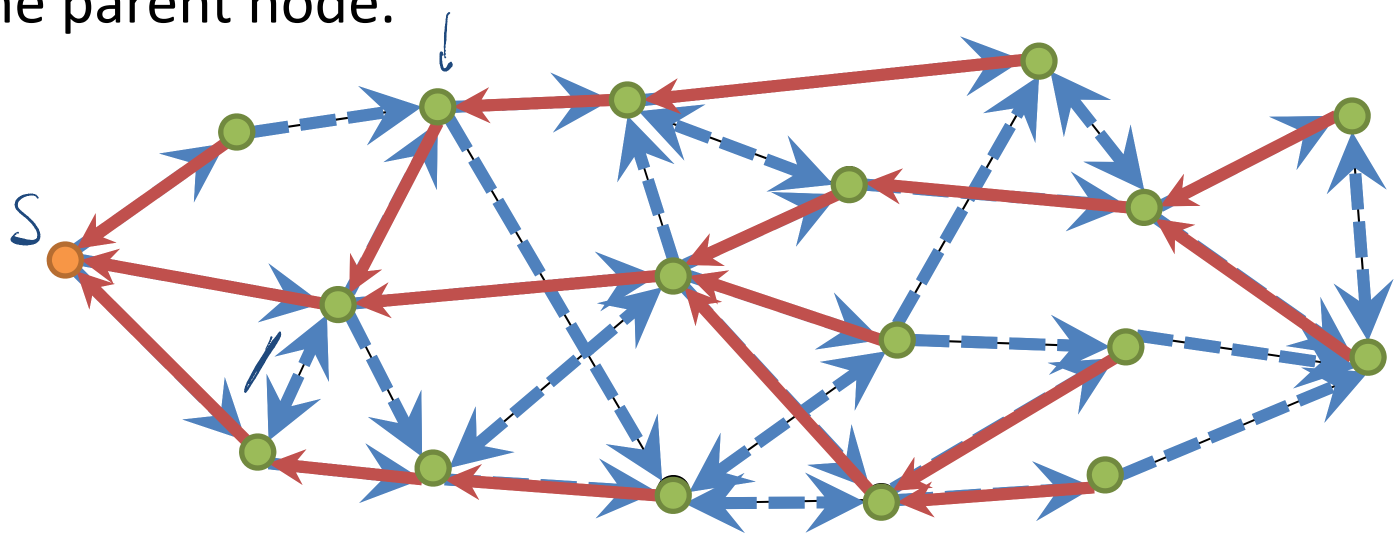
- on graph $G = (V, E)$

Flooding Spanning Tree

- The flooding algorithm can be used to compute a spanning tree of the network.

Idea:

- Source s is the root of the tree
- For all other nodes, neighbor from which M is received first is the parent node.



Flooding Spanning Tree Algorithm

Source node s:

initially do

parent $:= \bot$ // s is the root

send M to all neighbors

Non-source node u:

upon receiving M from some neighbor v

if M has not been received before **then**

parent $:= v$

send M to all neighbors except v

Spanning Tree: Synchronous Systems

- In tree: distance of v to root $=$ round in which v is reached
- In synchronous systems, a node v are reached in round r if and only if $dist_G(s, v) = r$

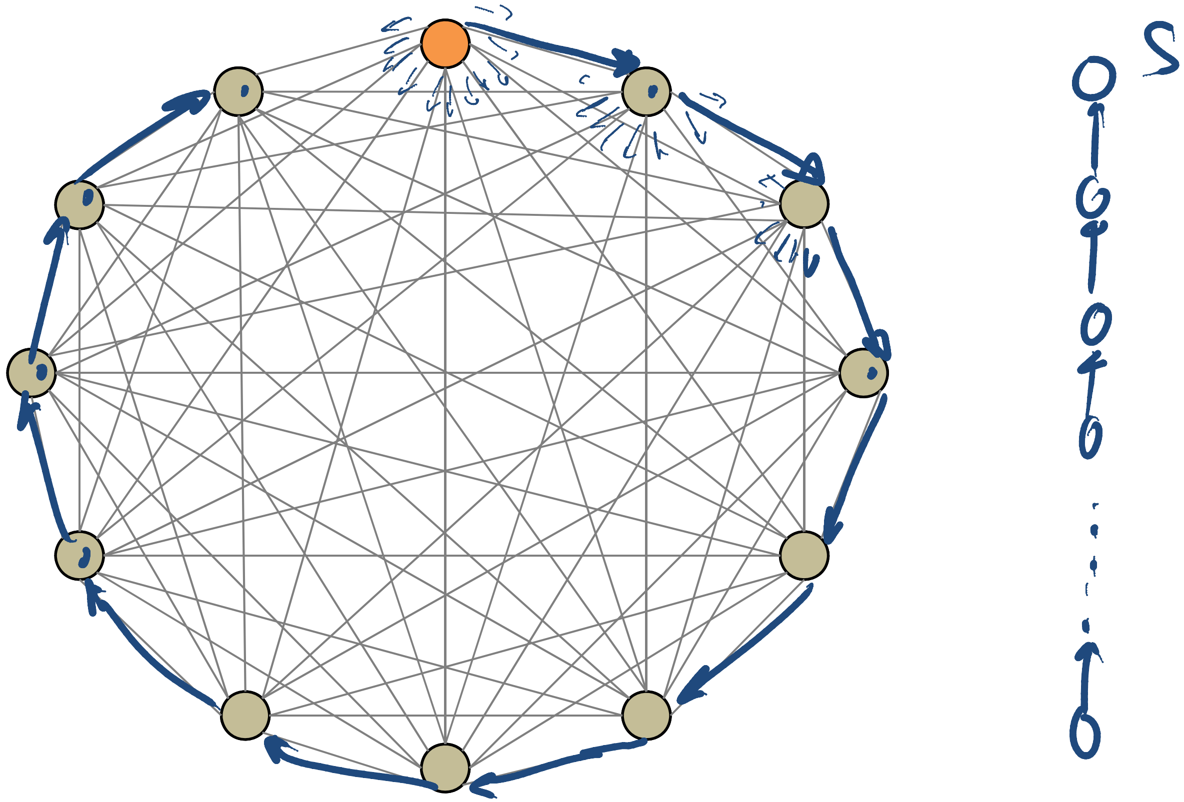
Shortest Path Tree = BFS Tree (BFS = breadth first search)

- tree which preserves graph distances to root node

Theorem: In synchronous systems, the flooding algorithm constructs a BFS tree.

Spanning Tree: Asynchronous Systems

How does the spanning tree look if comm. is asynchronous?

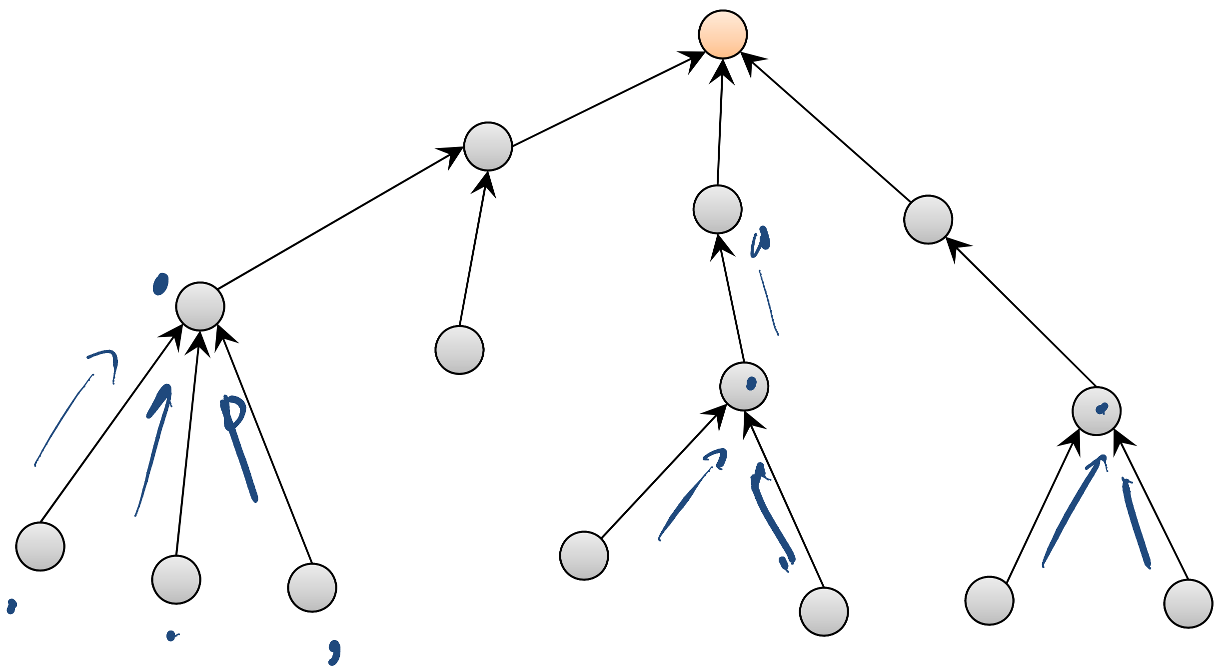


Observation: In asynchronous executions, the depth of the tree can be $n - 1$ even if the radius/diameter of the graph is 1.

Convergecast

- "Opposite" of broadcast
- Given a rooted spanning tree, communicate from all the nodes to the root

Example: Compute sum of values in a rooted tree



Convergecast Algorithm

Leaf node v:
initially do
send message to parent
(e.g., send input value)

Inner node u:
upon receiving message from child node v
if u has received messages from all children **then**
send message to parent
(e.g., send sum of all inputs in u's subtree)

Root node r:
upon receiving message from child node v
if r has received messages from all children **then**
convergecast terminates

Convergecast: Analysis & Remarks

Time Complexity:

depth of tree

Message Complexity:

#edges of tree = #nodes − 1

Application of the convergecast algorithm:

- Computing functions, e.g.:
  - min, max, sum, average, median, ...
- Termination detection
  - inform parent as soon as all nodes in subtree have terminated
- ...

Flooding/Echo Algorithm

- If a leader (root), but no spanning tree exists, flooding and convergecast can be used together for computing functions, ...

1. Use flooding to construct a tree
2. Use convergecast (echo) to report back to the root when done

Time Complexity of Flooding + Convergecast (Echo):

$O(\text{depth of constructed tree})$

synch: $O(D)$

asynch: $O(n)$

Constructing Good Trees

- When combining flooding and convergecast, the time complexity is linear in the depth of the constructed tree.
- In synchronous systems, the tree is a BFS tree (shortest path tree), i.e., the depth of the tree is $O\big(diam(G)\big)$
  - optimal time complexity: $O\big(diam(G)\big)$
- In asynchronous systems, the time complexity can be $\Omega(n)$, even if the graph has a very small diameter!
- Convergecast / low diameter spanning trees are important!
- How can be construct a BFS tree in an asynchronous system?

Constructing Shortest Path Tree

Dijkstra

- Grow tree from source s
- At intermediate step t, subtree of all nodes at distance $\leq r_t$ from source node s
- Next step: add node with min. distance to s

Bellman-Ford

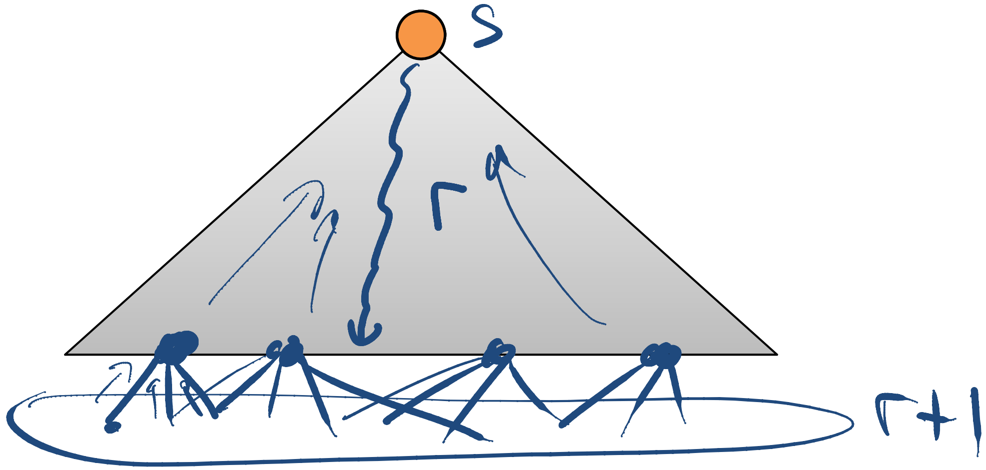
- Each node v keeps a distance estimate d_v to s
  - initially: $d_s = 0,\ d_v = \infty$ (for all $v \neq s$)
- In each step, all nodes update their estimate based on neighbor estimates:

$$d_v = \min\left\{d_v, \min_{u \in N(v)} \{d_u + 1\}\right\}$$

Distributed Dijkstra

- In our case, the graph is unweighted
- We can therefore grow the tree level by level
  - Essentially like in a synchronous execution

- Assume, the tree is constructed up to distance r from s
- How can we add the next level?



Distributed Dijkstra

- Source/root node coordinates the phases

Algorithm for Phase $r + 1$:

1. Root node broadcasts *"start phase* $r + 1$*"* in current tree
2. Leaf nodes (level r nodes) send *"join* $r + 1$*"* to neighbors
3. Node v receiving *"join* $r + 1$*"* from neighbor u:
   1. First such message: u becomes parent of v, v sends *ACK* to u
   2. Otherwise, v sends *NACK* to u
4. After receiving *ACK* or *NACK* from all neighbors, level r nodes report back to root by starting a convergecast
5. When the convergecast terminates at the root, the root can start the next phase

Distributed Dijkstra: Analysis

Time Complexity:

$$O\left(\sum_{i=1}^{D} i\right) = O(D^2)$$

Message Complexity: $O(m + n \cdot D)$

Distributed Bellman-Ford

Basic Idea:

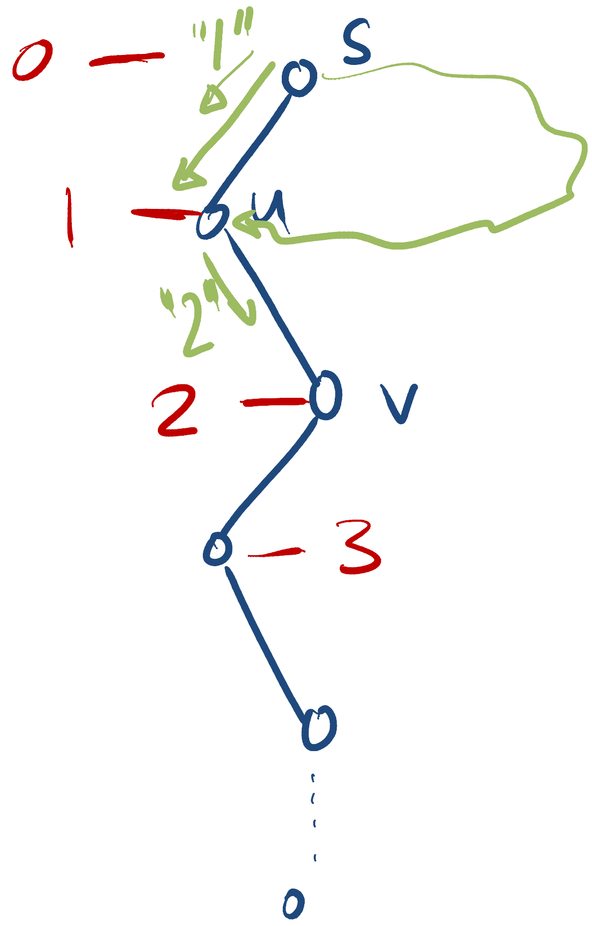
- Each node u stores an integer d_u with the current guess for the distance to the root node s
- Whenever a node u can improve d_u, u informs its neighbors

Algorithm:

1. Initialization: $d_s \coloneqq 0$, for $v \neq s$: $d_v \coloneqq \infty$, $\text{parent}_v \coloneqq \perp$
2. Root s sends "1" to all neigbors
3. For all other nodes u:
   upon receiving message "x" with $x < d_u$ from neighbor v **do**
      set $d_u \coloneqq x$
      set $\text{parent}_u \coloneqq v$
      send "$x + 1$" to all neighbors (except v)

Distr. Bellman-Ford: Time Complexity

Theorem: The time complexity of the distributed Bellman-Ford algorithms is $rad(G,s) = O\big(diam(G)\big)$.



Distr. Bellman-Ford: Message Complexity

Theorem: The message complexity of the distributed Bellman-Ford algorithms is $O(|E| \cdot |V|)$. (even if D=1)

$x \leq n-1$

Distributed BFS Tree Construction

Synchronous

- Time: $O\big(diam(G)\big)$,   Messages: $O(|E|)$
- both optimal

Asynchronous

- **Distributed Dijkstra:**
  Time: $O(diam(G)^2)$,   Messages: $O\big(|E| + |V| \cdot diam(G)\big)$
- **Distributed Bellman-Ford:**
  Time: $O\big(diam(G)\big)$,   Messages: $O(|E| \cdot |V|)$
- **Best known trade-off between time and messages:**
  Time: $O(diam(G) \cdot \log^3|V|)$, Messages: $O(|E| + |V| \cdot \log^3|V|)$
  – based on synchronizers   Awerbuch
    (generic way of translating synchronous algorithms into asynch. ones)

Synchronizers

Motivation:

- synchronous algorithms are often simpler and more efficient than asynchronous ones
- however, often real systems are asynchronous

Goal: Run synchronous algorithms in asynchronous systems

Synchronizer:

- Locally simulates rounds at all nodes
- Needs to make sure that when running a synchronous algorithm using the locally simulated rounds:

The local schedules are the same as in the synchronous exec.

Simple Local Synchronizer

Locally simulating rounds (node u):

- Node u generates clock pulses to start each new round
- Before starting round r, u needs to make sure that all messages of round $r - 1$ have been received.
- After starting round r, u sends all messages of round r

Making sure that all messages of current round are received:

- Need to know which neighbors want to send messages
- Easy if all neighbors send a message
- **Solution:**
  In each round, all nodes send a message to all neighbors
  - If the synch. algorithm does not send a message, send a dummy message instead

Simple Local Synchronizer

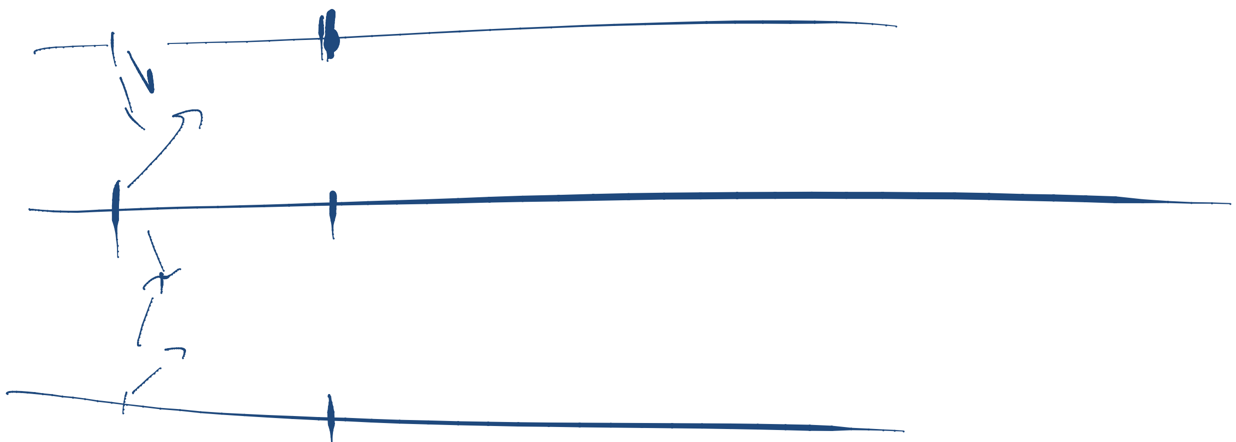
Simulate Round r:

1. Wait until round $r-1$ msg. from all neighbors are received
2. Send round r msg. to all neighbors
   - send dummy msg. to nodes to which no ordinary msg. is sent

Theorem: Algorithm correctly allows to run a synchronous alg. in an asynchronous system.

Simple Local Synchronizer

Theorem: In an asynchronous system, if all nodes start simulation at time 0, the time complexity to simulate R rounds is R.



Theorem: The total number of dummy messages to simulate R rounds is at most $O(R \cdot |E|)$.

Synchronizer S

Synchronizer Time Complexity $T(S)$:

- Time complexity for simulating one round

Synchronizer Message Complexity $M(S)$:

- Number of control messages for simulating one round

Simple Synchronizer:

- Time Complexity: 1 Message Complexity: $2|E|$

Other trade-offs between time and message complexity are possible, e.g.,

- $T(S) = O(\log|V|), \quad M(S) = O(|V|)$
- $T(S) = M(S) = O(\log^3|V|)$
- More details in the Network Algorithms lecture!

BFS Tree with Synchronizer

Synchronous BFS Tree Construction:

- Time Complexity: $O\big(diam(G)\big)$ Message Complexity: $O(|E|)$

Asynchronous BFS Tree Constr. Using Synchronizer S:

- Time Complexity: $O\big(diam(G) \cdot T(S)\big)$
- Msg. Complexity: $O\big(|E| + diam(G) \cdot M(S)\big)$

With Simple Synchronizer:

- Time Compl.: $O\big(diam(G)\big)$ Msg. Compl.: $O(diam(G) \cdot |E|)$
- Slightly better than distributed Bellman-Ford

- Best BFS algorithm is based on best known synchronizer

Leader Election

Task: Each node has an input value, compute sum of values

Solution: Compute spanning tree and use convergecast on spanning tree (i.e., flooding + convergecast)

Problem: What if we don't have a source/root node?

We need to choose a root node

- known as the *leader election problem*

Solving leader election:

- E.g.: Choose node with smallest ID
- How to find node with smallest ID?

Solving Leader Election

Choose node with smallest ID

Algorithm for node u:

- Node u stores smallest known ID in variable x_u

1. Initially, u sets $x_u \coloneqq \mathrm{ID}_u$ and sends x_u to all neighbors
2. when receiving $x_v < x_u$ from neighbor v:
   $x_u \coloneqq x_v$
   send x_u to all neighbors (except v)

Time Complexity:

Solving Leader Election

Choose node with smallest ID

Algorithm for node u:

- Node u stores smallest known ID in variable x_u

1. Initially, u sets $x_u \coloneqq \mathrm{ID}_u$ and sends x_u to all neighbors
2. when receiving $x_v < x_u$ from neighbor v:

   $x_u \coloneqq x_v$

   send x_u to all neighbors (except v)

Message Complexity:

Leader Election

Simple leader election algorithm has time complexity $O\big(diam(G)\big)$ and message complexity $O(|V| \cdot |E|)$.

Problems:

- While time compl. is optimal, msg. complexity is extremely high
- It is not clear when/how to terminate

- Like for BFS tree construction, there are many possible trade-offs between time and message complexity, e.g.:
  - Time Complexity: $O(|V|)$, Message Complexity: $O(|E| + |V| \cdot \log|V|)$
- Termination can be solved (at some cost)
- More on leader election: Network Algorithms Lecture