This Week

• **Synchronous distributed algorithms:**
  – Leader Election
  – Maximal Independent Set
  – Breadth-First Spanning Trees
  – Shortest Paths Trees (started)
  – Shortest Paths Trees (finish)

• **Asynchronous distributed algorithms:**
  – Breadth-First Spanning Trees
  – Shortest Paths Trees
Distributed Networks

• Based on undirected graph $G = (V, E)$.
  – $n = |V|$
  – $\Gamma(u)$, set of neighbors of vertex $u$.
  – $\text{deg}(u) = |\Gamma(u)|$, number of neighbors of vertex $u$.

• Associate a process with each graph vertex.

• Associate two directed communication channels with each edge.
Synchronous Distributed Algorithms
Synchronous Network Model

• Processes at graph vertices, communicate using messages.
• Each process has output ports, input ports that connect to communication channels.

• Algorithm executes in synchronous rounds.
• In each round:
  – Each process sends messages on its ports.
  – Each message gets put into the channel, delivered to the process at the other end.
  – Each process computes a new state based on the arriving messages.
Leader Election
**n-vertex Clique**

- **Theorem:** There is no algorithm consisting of deterministic, indistinguishable processes that is guaranteed to elect a leader in $G$.

- **Theorem:** There is an algorithm consisting of deterministic processes with UIDs that is guaranteed to elect a leader.
  - 1 round, $n^2$ messages.

- **Theorem:** There is an algorithm consisting of randomized, indistinguishable processes that eventually elects a leader, with probability 1.
  - Expected time $\leq \frac{1}{1-\epsilon}$.
  - With probability $\geq 1 - \epsilon$, finishes in one round.
Maximal Independent Set (MIS)
MIS

- **Independent**: No two neighbors are both in the set.
- **Maximal**: We can’t add any more nodes without violating independence.
- Every node is either in $S$ or has a neighbor in $S$.
- **Assume**:
  - No UIDs
  - Processes know a good upper bound on $n$.
- **Require**:
  - Compute an MIS $S$ of the network graph.
  - Each process in $S$ should output *in*, others output *out*.
Luby’s Algorithm

• Initially all nodes are active.
• At each phase, some active nodes decide to be in, others decide to be out, the rest continue to the next phase.

• Behavior of active node at a phase:
  • Round 1:
    – Choose a random value $r$ in $\{1, 2, \ldots, n^5\}$, send it to all neighbors.
    – Receive values from all active neighbors.
    – If $r$ is strictly greater than all received values, then join the MIS, output in.
  • Round 2:
    – If you joined the MIS, announce it in messages to all (active) neighbors.
    – If you receive such an announcement, decide not to join the MIS, output out.
    – If you decided one way or the other at this phase, become inactive.
Luby’s Algorithm

• **Theorem:** If Luby’s algorithm ever terminates, then the final set $S$ is an MIS.

• **Theorem:** With probability at least $1 - \frac{1}{n}$, all nodes decide within $4 \log n$ phases.
Breadth-First Spanning Trees
Breadth-First Spanning Trees

• Distinguished vertex \( v_0 \).

• Processes must produce a Breadth-First Spanning Tree rooted at vertex \( v_0 \).

• Assume:
  – UIDs.
  – Processes have no knowledge about the graph.

• Output: Each process \( i \neq i_0 \) should output \( \text{parent}(j) \).
Simple BFS Algorithm

- Processes mark themselves as they get incorporated into the tree.
- Initially, only $i_0$ is marked.
- **Algorithm for process $i$:**
  - **Round 1:**
    - If $i = i_0$ then process $i$ sends a *search* message to its neighbors.
    - If process $i$ receives a message, then it:
      - Marks itself.
      - Selects $i_0$ as its parent, outputs $parent(i_0)$.
      - Plans to send at the next round.
  - **Round $r > 1$:**
    - If process $i$ planned to send, then it sends a *search* message to its neighbors.
    - If process $i$ is not marked and receives a message, then it:
      - Marks itself.
      - Selects one sending neighbor, $j$, as its parent, outputs $parent(j)$.
      - Plans to send at the next round.
Correctness

- State variables, per process:
  - marked, a Boolean, initially true for $i_0$, false for others
  - parent, a UID or undefined
  - send, a Boolean, initially true for $i_0$, false for others
  - uid

- Invariants:
  - At the end of $r$ rounds, exactly the processes at distance $\leq r$ from $v_0$ are marked.
  - A process $\neq i_0$ has its parent defined iff it is marked.
  - For any process at distance $d$ from $v_0$, if its parent is defined, then it is the UID of a process at distance $d - 1$ from $v_0$. 
Complexity

- **Time complexity:**
  - Number of rounds until all nodes outputs their parent information.
  - Maximum distance of any node from $v_0$, which is $\leq diam$

- **Message complexity:**
  - Number of messages sent by all processes during the entire execution.
  - $O(|E|)$
Bells and Whistles

• Child pointers:
  – Send *parent/nonparent* responses to search messages.

• Distances:
  – Piggyback distances on *search* messages.

• Termination:
  – Convergecast starting from the leaves.

• Applications:
  – Message broadcast from the root
  – Global computation
Shortest Paths Trees
Shortest Paths

- Generalize the BFS problem to allow weights on the graph edges, \( \text{weight}_{\{u,v\}} \) for edge \( \{u, v\} \)
- Connected graph \( G = (V, E) \), root vertex \( v_0 \), process \( i_0 \).
- Processes have UIDs.
- Processes know their neighbors and the weights of their incident edges, but otherwise have no knowledge about the graph.
Shortest Paths

• Processes must produce a Shortest-Paths Spanning Tree rooted at vertex \( v_0 \).

• Branches are directed paths from \( v_0 \).
  – **Spanning:** Branches reach all vertices.
  – **Shortest paths:** The total weight of the tree branch to each node is the minimum total weight for any path from \( v_0 \) in \( G \).

• **Output:** Each process \( i \neq i_0 \) should output\( parent(j), distance(d) \), meaning that:
  – \( j \)'s vertex is the parent of \( i \)'s vertex on a shortest path from \( v_0 \),
  – \( d \) is the total weight of a shortest path from \( v_0 \) to \( j \).
Bellman-Ford Shortest Paths Algorithm

- **State variables:**
  - $dist$, a nonnegative real or $\infty$, representing the shortest known distance from $v_0$. Initially 0 for process $i_0$, $\infty$ for the others.
  - $parent$, a UID or undefined, initially undefined.
  - $uid$

- **Algorithm for process $i$:**
  - At each round:
    - Send a $distance(dist)$ message to all neighbors.
    - Receive messages from neighbors; let $d_j$ be the distance received from neighbor $j$.
    - Perform a relaxation step:
      $$dist := \min(dist, \min_j (d_j + weight_{ij})).$$
    - If $dist$ decreases then set $parent := j$, where $j$ is any neighbor that produced the new $dist$. 
Correctness

- **Claim:** Eventually, every process $i$ has:
  - $dist =$ minimum weight of a path from $i_0$ to $i$, and
  - if $i \neq i_0$, $parent =$ the previous node on some shortest path from $i_0$ to $i$.

- **Key invariant:**
  - For every $r$, at the end of $r$ rounds, every process $i \neq i_0$ has its $dist$ and $parent$ corresponding to a shortest path from $i_0$ to $i$ among those paths that consist of at most $r$ edges; if there is no such path, then $dist = \infty$ and $parent$ is undefined.
Complexity

- **Time complexity:**
  - Number of rounds until all the variables stabilize to their final values.
  - $n - 1$ rounds

- **Message complexity:**
  - Number of messages sent by all processes during the entire execution.
  - $O(n \cdot |E|)$

- More expensive than BFS:
  - $diam$ rounds,
  - $O(|E|)$ messages

- **Q:** Does the time bound really depend on $n$?
Child Pointers

• Ignore repeated messages.
• When process $i$ receives a message that it does not use to improve $\text{dist}$, it responds with a nonparent message.
• When process $i$ receives a message that it uses to improve $\text{dist}$, it responds with a parent message, and also responds to any previous parent with a nonparent message.
• Process $i$ records nodes from which it receives parent messages in a set $\text{children}$.
• When process $i$ receives a nonparent message from a current child, it removes the sender from its $\text{children}$.
• When process $i$ improves $\text{dist}$, it empties $\text{children}$.
Termination

• **Q:** How can the processes learn when the shortest-paths tree is completed?

• **Q:** How can a process even know when it can output its own *parent* and *distance*?

• If processes knew an upper bound on \( n \), then they could simply wait until that number of rounds have passed.

• But what if they don’t know anything about the graph?

• Recall termination for BFS: Used *convergecast*.

• **Q:** Does that work here?
Termination

• **Q:** How can the processes learn when the shortest-paths tree is completed?
• **Q:** Does convergecast work here?
• Yes, but it’s trickier, since the tree structure changes.

• **Key ideas:**
  – A process $\neq i_0$ can send a *done* message to its current parent after:
    • It has received responses to all its *distance* messages, so it believes it knows who its children are, and
    • It has received *done* messages from all of those children.
  – The same process may be involved several times in the convergecast, based on improved estimates.
Asynchronous Distributed Algorithms
Asynchronous Network Model

• Complications so far:
  – Processes act concurrently.
  – A little nondeterminism.

• Now things get much worse:
  – No rounds---process steps and message deliveries happen at arbitrary times, in arbitrary orders.
  – Processes get out of synch.
  – Much more nondeterminism.

• Understanding asynchronous distributed algorithms is hard because we can’t understand exactly how they execute.

• Instead, we must understand abstract properties of executions.
Aynchronous Network Model

- Lynch, Distributed Algorithms, Chapter 8.
- **Processes** at nodes of an undirected graph $G = (V, E)$, communicate using messages.
- **Communication channels** associated with edges (one in each direction on each edge).
  - $C_{u,v}$, channel from vertex $u$ to vertex $v$.
- Each process has **output ports and input ports** that connect it to its communication channels.
- Processes need not be distinguishable.
Channel Automaton $C_{u,v}$

- Formally, an input/output automaton.
- Input actions: $send(m)_{u,v}$
- Output actions: $receive(m)_{u,v}$
- State variable:
  - $mqueue$, a FIFO queue, initially empty.
- Transitions:
  - $send(m)_{u,v}$
    - Effect: add $m$ to $mqueue$.
  - $receive(m)_{u,v}$
    - Precondition: $m = head(mqueue)$
    - Effect: remove head of $mqueue$
Process Automaton $P_u$

- Associate a process automaton with each vertex of $G$.
- To simplify notation, let $P_u$ denote the process automaton at vertex $u$.
  - But the process does not “know” $u$.
- $P_u$ has $send(m)_{u,v}$ outputs and $receive(m)_{v,u}$ inputs.
- May also have external inputs and outputs.
- Has state variables.
- Keeps taking steps (eventually).
Example: \( \text{Max}_u \) Process Automaton

- Input actions: \( \text{receive}(m)_{v,u} \)
- Output actions: \( \text{send}(m)_{u,v} \)
- State variables:
  - \( \text{max} \), a natural number, initially \( x_u \)
  - For each neighbor \( v \):
    - \( \text{send}(v) \), a Boolean, initially \( \text{true} \)
- Transitions:
  - \( \text{receive}(m)_{v,u} \)
    - Effect: if \( m > \text{max} \) then
      - \( \text{max} := m \)
      - for every \( w \), \( \text{send}(w) := \text{true} \)
  - \( \text{send}(m)_{u,v} \)
    - Precondition: \( \text{send}(v) = \text{true} \) and \( m = \text{max} \)
    - Effect: \( \text{send}(v) := \text{false} \)
Combining Processes and Channels

• Undirected graph $G = (V, E)$.
• Process $P_u$ at each vertex $u$.
• Channels $C_{u,v}$ and $C_{v,u}$, associated with each edge $\{u, v\}$.
  • $send(m)_{u,v}$ output of process $P_u$ gets identified with $send(m)_{u,v}$ input of channel $C_{u,v}$.
  • $receive(m)_{v,u}$ output of channel $C_{v,u}$ gets identified with $receive(m)_{v,u}$ input of process $P_u$.
• Steps involving such a shared action involve simultaneous state transitions for a process and a channel.
Execution

- No synchronous rounds anymore.
- The system executes by performing enabled steps, one at a time, in any order.
- Formally, an execution is modeled as a sequence of individual steps.
- Different from the synchronous model, in which all processes take steps concurrently at each round.

Assume enabled steps eventually occur:
- Each channel always eventually delivers the first message in its queue.
- Each process always eventually performs some enabled step.
Combining $\text{Max}$ Processes and Channels

- Each process $\text{Max}_u$ starts with an initial value $x_u$.
- They all send out their initial values, and propagate their max values, until everyone has the globally-maximum value.
- Sending and receiving steps can happen in many different orders, but in all cases the global max will eventually arrive everywhere.
Max System
Max System

Graph showing connections between nodes labeled 3, 4, 5, 7, and 10.

Nodes are connected with arrows indicating direction of influence or flow.
Max System
Max System

Diagram showing a network of nodes labeled 5, 4, 10, and 7, connected by arrows indicating the direction of flow with values 10, 7, and 41.
Max System
Max System
Max System
Max System
Complexity

- **Message complexity:**
  - Number of messages sent by all processes during the entire execution.
  - $O(n \cdot |E|)$

- **Time complexity:**
  - **Q:** What should we measure?
  - Not obvious, because the various components are taking steps in arbitrary orders—no “rounds”.
  - **A common approach:**
    - Assume real-time upper bounds on the time to perform basic steps:
      - $d$ for a channel to deliver the next message, and
      - $l$ for a process to perform its next step.
    - Infer a real-time upper bound for solving the overall problem.
Complexity

- **Time complexity:**
  - Assume real-time upper bounds on the time to perform basic steps:
    - $d$ for a channel to deliver the next message, and
    - $l$ for a process to perform its next step.
  - Infer a real-time upper bound for solving the problem.

- **For the Max system:**
  - Ignore local processing time ($l = 0$), consider only channel sending time.
  - Straightforward upper bound: $O(diam \cdot n \cdot d)$
    - Consider the time for the max to reach any particular vertex $u$, along a shortest path in the graph.
    - At worst, it waits in each channel on the path for every other value, which is at most time $n \cdot d$ for that channel.
Breadth-First Spanning Trees
Breadth-First Spanning Trees

• **Problem:** Compute a Breadth-First Spanning Tree in an asynchronous network.
• Connected graph $G = (V, E)$.
• Distinguished root vertex $v_0$.
• Processes have no knowledge about the graph.
• Processes have UIDs
  – $i_0$ is the UID of the root $v_0$.
  – Processes know UIDs of their neighbors, and know which ports are connected to each neighbor.
• Processes must produce a BFS tree rooted at $v_0$.
• Each process $i \neq i_0$ should output $parent(j)$, meaning that $j$’s vertex is the parent of $i$’s vertex in the BFS tree.
First Attempt

• Just run the simple synchronous BFS algorithm asynchronously.
• Process $i_0$ sends search messages, which everyone propagates the first time they receive it.
• Everyone picks the first node from which it receives a search message as its parent.

• Nondeterminism:
  – No longer any nondeterminism in process decisions.
  – But plenty of new nondeterminism: orders of message deliveries and process steps.
Process Automaton $P_u$

• Input actions: $\text{receive}(\text{search})_{v,u}$
• Output actions: $\text{send}(\text{search})_{u,v}; \text{parent}(v)_u$
• State variables:
  – $\text{parent}$: $\Gamma(u) \cup \{\bot\}$, initially $\bot$
  – $\text{reported}$: Boolean, initially false
  – For every $v \in \Gamma(u)$:
    • $\text{send}(v) \in \{\text{search}, \bot\}$, initially $\text{search if } u = v_0$, else $\bot$

• Transitions:
  – $\text{receive}(\text{search})_{v,u}$
    • Effect: if $u \neq v_0$ and $\text{parent} = \bot$ then
      – $\text{parent} := v$
      – for every $w$, $\text{send}(w) := \text{search}$
Process Automaton $P_u$

• Transitions:
  
  – $receive(search)_{v,u}$
    • Effect: if $u \neq v_0$ and $parent = \bot$ then
      – $parent := v$
      – for every $w$, $send(w) := search$
  
  – $send(search)_{u,v}$
    • Precondition: $send(v) = search$
    • Effect: $send(v) := \bot$
  
  – $parent(v)_u$
    • Precondition: $parent = v$ and $reported = false$
    • Effect: $reported := true$
Running Simple BFS Asynchronously
Final Spanning Tree
Actual BFS
Anomaly

- Paths produced by the algorithm may be longer than the shortest paths.
- Because in asynchronous networks, messages may propagate faster along longer paths.
Complexity

- **Message complexity:**
  - Number of messages sent by all processes during the entire execution.
  - $O(|E|)$

- **Time complexity:**
  - Time until all processes have chosen their parents.
  - Neglect local processing time.
  - $O(\text{diam} \cdot d)$
  - **Q:** Why \text{diam}, when some of the paths are longer?
  - The time until a node receives a \textit{search} message is at most the time it would take on a shortest path.
Extensions

• Child pointers:
  – As for synchronous BFS.
  – Everyone who receives a search message sends back a parent or nonparent response.

• Termination:
  – After a node has received responses to all its search messages, it knows who its children are, and knows they are marked.
  – The leaves of the tree learn who they are.
  – Use a convergecast strategy, as before.
  – Time complexity: After the tree is done, it takes time $O(n \cdot d)$ for the done information to reach $i_0$.
  – Message complexity: $O(n)$
Applications

- **Message broadcast:**
  - Process $i_0$ can use the tree (with child pointers) to broadcast a message.
  - Takes $O(n \cdot d)$ time and $n$ messages.

- **Global computation:**
  - Suppose every process starts with some initial value, and process $i_0$ should determine the value of some function of the set of all processes’ values.
  - Use convergecast on the tree.
  - Takes $O(n \cdot d)$ time and $n$ messages.
Second Attempt

• A relaxation algorithm, like synchronous Bellman-Ford.
• Before, we corrected for paths with many hops but low weights.
• Now, instead, correct for errors caused by asynchrony.
• **Strategy:**
  – Each process keeps track of the hop distance, changes its parent when it learns of a shorter path, and propagates the improved distances.
  – Eventually stabilizes to a breadth-first spanning tree.
Process Automaton $P_u$

- **Input actions:** $receive(m)_{v,u}$, $m$ a nonnegative integer
- **Output actions:** $send(m)_{u,v}$, $m$ a nonnegative integer

- **State variables:**
  - $parent$: $\Gamma(u) \cup \{ \bot \}$, initially $\bot$
  - $dist \in \mathbb{N} \cup \{\infty\}$, initially $0$ if $u = v_0$, $\infty$ otherwise
    - For every $v \in \Gamma(u)$:
      - $send(v)$, a FIFO queue of $\mathbb{N}$, initially $(0)$ if $u = v_0$, else empty

- **Transitions:**
  - $receive(m)_{v,u}$
    - Effect: if $m + 1 < dist$ then
      - $dist := m + 1$
      - $parent := v$
      - for every $w$, add $dist$ to $send(w)$
Process Automaton $P_u$

• Transitions:
  – $receive(m)_{v,u}$
    • Effect: if $m + 1 < dist$ then
      – $dist := m + 1$
      – $parent := v$
      – for every $w$, add $m + 1$ to $send(w)$
  – $send(m)_{u,v}$
    • Precondition: $m = \text{head}(send(v))$
    • Effect: remove head of $send(v)$

• No terminating actions...
Correctness

• For synchronous BFS, we characterized precisely the situation after $r$ rounds.
• We can’t do that now.
• Instead, state abstract properties, e.g., invariants and timing properties, e.g.:
  • **Invariant:** At any point, for any node $u \neq v_0$, if its $dist \neq \infty$, then it is the actual distance on some path from $v_0$ to $u$, and its *parent* is $u$’s predecessor on such a path.
  • **Timing property:** For any node $u$, and any $r$, $0 \leq r \leq diam$, if there is an at-most-$r$-hop path from $v_0$ to $u$, then by time $r \cdot n \cdot d$, node $u$’s $dist$ is $\leq r$. 
Complexity

- **Message complexity:**
  - Number of messages sent by all processes during the entire execution.
  - $O(n \mid E\mid)$

- **Time complexity:**
  - Time until all processes’ `dist` and `parent` values have stabilized.
  - Neglect local processing time.
  - $O(diam \cdot n \cdot d)$
    - Time until each node receives a message along a shortest path, counting time $O(n \cdot d)$ to traverse each link.
Termination

• **Q:** How can processes learn when the tree is completed?
• **Q:** How can a process know when it can output its own `dist` and `parent`?
• Knowing a bound on `n` doesn’t help here: can’t use it to count rounds.

• Can use **convergecast**, as for synchronous Bellman-Ford:
  – Compute and recompute child pointers.
  – Process `≠ v₀` sends `done` to its current parent after:
    • It has received responses to all its messages, so it believes it knows all its children, and
    • It has received `done` messages from all of those children.
  – The same process may be involved several times, based on improved estimates.
Uses of Breadth-First Spanning Trees

• Same as in synchronous networks, e.g.:
  – Broadcast a sequence of messages
  – Global function computation

• Similar costs, but now count time $d$ instead of one round.
Shortest Paths Trees
Shortest Paths

• **Problem:** Compute a *Shortest Paths Spanning Tree* in an asynchronous network.

• Connected weighted graph, root vertex \( v_0 \).

• \( \text{weight}_{\{u,v\}} \) for edge \( \{u, v\} \).

• Processes have no knowledge about the graph, except for weights of incident edges.

• UIDs

• Processes must produce a Shortest Paths spanning tree rooted at \( v_0 \).

• Each process \( u \neq v_0 \) should output its distance and parent in the tree.
Shortest Paths

• Use a relaxation algorithm, once again.
• Asynchronous Bellman-Ford.

• Now, it handles two kinds of corrections:
  – Because of long, small-weight paths (as in synchronous Bellman-Ford).
  – Because of asynchrony (as in asynchronous Breadth-First search).

• The combination leads to surprisingly high message and time complexity, much worse than either type of correction alone (exponential).
Asynch Bellman-Ford, Process $P_u$

- **Input actions**: $receive(m)_{v,u}$, $m$ a nonnegative integer
- **Output actions**: $send(m)_{u,v}$, $m$ a nonnegative integer

- **State variables**:
  - $parent$: $\Gamma(u) \cup \{\perp\}$, initially $\perp$
  - $dist \in N \cup \{\infty\}$, initially 0 if $u = v_0$, $\infty$ otherwise
  - For every $v \in \Gamma(u)$:
    - $send(v)$, a FIFO queue of $N$, initially $(0)$ if $u = v_0$, else empty

- **Transitions**:
  - $receive(m)_{v,u}$
    - **Effect**: if $m + weight_{v,u} < dist$ then
      - $dist := m + weight_{v,u}$
      - $parent := v$
      - for every $w$, add $dist$ to $send(w)$
Asynch Bellman-Ford, Process $P_u$

• Transitions:
  – $receive(m)_{v,u}$
    • Effect: if $m + weight_{v,u} < dist$ then
      – $dist := m + weight_{v,u}$
      – $parent := v$
      – for every $w$, add $dist$ to $send(w)$
  – $send(m)_{u,v}$
    • Precondition: $m = head(send(v))$
    • Effect: remove head of $send(v)$

• No terminating actions...
Correctness:
Invariants and Timing Properties

- **Invariant:** At any point, for any node $u \neq v_0$, if its $dist \neq \infty$, then it is the actual distance on some path from $v_0$ to $u$, and its $parent$ is $u$’s predecessor on such a path.

- **Timing property:** For any node $u$, and any $r$, $0 \leq r \leq diam$, if $p$ is any at-most-$r$-hop path from $v_0$ to $u$, then by time $???, node u$’s $dist$ is $\leq$ total weight of $p$.

- **Q:** What is $???$?
  - It depends on how many messages might pile up in a channel.
  - This can be a lot!
Complexity

- $O(n!)$ simple paths from $v_0$ to any other node $u$, which is $O(n^n)$.
- So the number of messages sent on any channel is $O(n^n)$.
- Message complexity: $O(n^n |E|)$.
- Time complexity: $O(n^n \cdot n \cdot d)$.

Q: Are such exponential bounds really achievable?
Complexity

- **Q:** Are such exponential bounds really achievable?
- **Example:**
  - There is an execution of the network below in which node $v_k$ sends $2^k \approx 2^{n/2}$ messages to node $v_{k+1}$.
  - Message complexity is $\Omega(2^{n/2})$.
  - Time complexity is $\Omega(2^{n/2} d)$.
Complexity

- Execution in which node $v_k$ sends $2^k$ messages to node $v_{k+1}$.
- Possible distance estimates for $v_k$ are $2^k - 1, 2^k - 2, \ldots, 0$.
- Moreover, $v_k$ can take on all these estimates in sequence:
  - First, messages traverse upper links, $2^k - 1$.
  - Then last lower message arrives at $v_k$, $2^k - 2$.
  - Then lower message $v_{k-2} \rightarrow v_{k-1}$ arrives, reduces $v_{k-1}$’s estimate by 2, message $v_{k-1} \rightarrow v_k$ arrives on upper links, $2^k - 3$.
  - Etc. Count down in binary.
  - If this happens quickly, get pileup of $2^k$ search messages in $C_{k,k+1}$.
Termination

• Q: How can processes learn when the tree is completed?
• Q: How can a process know when it can output its own dist and parent?

• Convergecast, once again
  – Compute and recompute child pointers.
  – Process $\neq v_0$ sends done to its current parent after:
    • It has received responses to all its messages, so it believes it knows all its children, and
    • It has received done messages from all of those children.
  – The same process may be involved several (many) times, based on improved estimates.
Shortest Paths

• Moral: Unrestrained asynchrony can cause problems.

• What to do?

• Find out in 6.852/18.437, Distributed Algorithms!
What’s Next?

• 6.852/18.437 Distributed Algorithms
• Basic grad course
• Covers synchronous, asynchronous, and timing-based algorithms

• Synchronous algorithms:
  – Leader election
  – Building various kinds of spanning trees
  – Maximal Independent Sets and other network structures
  – Fault tolerance
  – Fault-tolerant consensus, commit, and related problems
Asynchronous Algorithms

• Asynchronous network model
• Leader election, network structures.
• Algorithm design techniques:
  – Synchronizers
  – Logical time
  – Global snapshots, stable property detection.
• Asynchronous shared-memory model
• Mutual exclusion, resource allocation

• Fault tolerance
• Fault-tolerant consensus and related problems
• Atomic data objects, atomic snapshots
• Transformations between models.
• Self-stabilizing algorithms
And More

• Timing-based algorithms
  – Models
  – Revisit some problems
  – New problems, like clock synchronization.

• Newer work (maybe):
  – Dynamic network algorithms
  – Wireless networks
  – Insect colony algorithms and other biological distributed algorithms