Multicast Communication

- **Multicast** is the delivery of a message to a group of receivers simultaneously in a single transmission from the source
- We will study two multicast approaches
  - Application-level multicasting
  - Epidemic algorithms

Application-Level Multicasting

- Processes in a peer-to-peer system are organized into an overlay network; data is disseminated along a multicast tree created on the overlay network
  - Sender is the root of the tree
  - The tree spans all the receivers
- A connection between two nodes in the overlay network may cross several physical links → routing messages within the overlay may not be optimal in comparison to network-level multicast
  - In network-level multicast, routers maintain multicast trees and forward multicast messages along the trees
When A sends a multicast message to B, C, and D using a tree in the overlay network (black), cost = cost(A-B) + cost(B-D) + cost(D-C) = 9 + 24 + 7 = 40.

When A sends a multicast message to B, C, and D using a tree over the physical network (red), cost = 1 + 7 + 1 + 10 + 1 + 5 + 1 = 26

Multicast Tree Construction in Chord

- Let S be the initiator of a multicast session
- S generates a (randomly chosen) multicast identifier mid, then looks up succ(mid) and promotes it to be the root of the multicast tree
- If node P wants to join the multicast tree, it executes Lookup(mid) to send a join request to the root
  - P becomes a forwarder in the tree
- When the join request arrives at a node Q
  - If Q has not seen a join request for mid before, it becomes a forwarder and P becomes the child of Q. Q will continue to forward the join request to the root.
  - If Q is already a forwarder for mid, becomes the child of Q and there is no need to forward the join request anymore.
- Sending a multicast message:
  - Sender sends the message toward the root by executing Lookup(mid)
  - The message is then sent along the tree
Epidemic Algorithms

• In large-scale distributed systems, epidemic algorithms are used to rapidly propagate information among a large collection of nodes with no central coordinator

• Assumptions
  • All updates for a specific data item are initiated at a single node
  • We can distinguish old data from new data because data is timestamped or versioned

• Terminology:
  – A node is called infected if it holds an update that it is willing to spread to other nodes
  – A node is called susceptible if it has not yet been updated
  – A node is called removed if it is not willing or able to spread its update

• Basic idea:
  – When a node is updated, it tries to “infect” other nodes as quickly as possible using pair-wise exchange of updates (like pair-wise spreading of a disease)
  – Eventually, each update should reach every node

Propagation Models

• Anti-entropy: each node regularly chooses another node at random, and exchanges updates with that node

• Gossiping: A node that has just been updated (i.e., infected) tells a number of other nodes about its update (i.e., infecting them)
Anti-Entropy

- A node P picks another node Q at random and exchanges updates with Q.
- Three approaches to exchanging updates:
  - Push: P only sends its updates to Q.
  - Pull: P only gets updates from Q.
  - Push-Pull: P and Q exchange updates (after which they hold the same information).
- A pure push-based or pull-based approach does not help spread updates quickly:
  - When a small number of nodes are infected, push-based approach is better.
  - When a large number of nodes are infected, pull-based approach is better.
- Push-pull is the best strategy: it takes $O(\log(N))$ rounds to disseminate an update to all N nodes (round = when every node has taken the initiative to exchange updates with another node).

Gossiping

- When a node P receives an update, it tries to push the update to an arbitrary other node Q.
- If Q was already updated by another node, P stops spreading the update (i.e., becomes removed) with probability $1/k$.
- Gossiping can rapidly spread updates, but cannot guarantee that all nodes will be updated:
  - When there is a large number of nodes, the fraction $s$ of nodes that will remain susceptible satisfy the equation $s = e^{(k+1)(1-s)}$.
    - Example: when $k=4$, $s < 0.007$.
- Combining anti-entropy with gossiping will ensure all nodes are eventually updated.
The relation between the fraction $s$ of susceptible nodes and the parameter $k$ in pure gossiping. The graph displays $\ln(s)$ as a function of $k$.

---

**Removing Data**

- Epidemic algorithms are excellent for spreading updates, but deletion of data items is hard
  - When a node deletes a data item, and then receives an old copy of the data item, the old copy will be interpreted as something new
    - The node can't distinguish between a deleted copy and no copy!
- Solution is to use death certificates
  - Treat deletes as updates and spread a death certificate
    - Every node keeps a record of the deletion using death certificate
    - Dormant death certificates should eventually be cleaned up
Removing Death Certificates

• A death certificate is timestamped when it is created
• Assuming death certificates propagate to all nodes in finite time, death certificates can be removed after this maximum propagation time has elapsed
• To provide hard guarantee that deletions are spread to all nodes, a few nodes maintain dormant death certificates that are never thrown away
  – Suppose node P has a dormant death certificate for data item x. If P receives an obsolete update for x, p will spread the death certificate for x again.

Information Aggregation Using Epidemic Algorithms

• Let every node i maintains an initial value $x_i$
• When node i contacts node j, they each update their value to $(x_i + x_j)/2$
• In the end each node will have computed the average $\bar{x} = \sum x_i / N$, where N is the number of nodes
• What happens if initially $x_i = 1$ if i=1 and $x_i = 0$ if i>1?
  – Eventually each node will compute the average (i.e., $x_i=1/N$), so every node can estimate the size of the system as being $1/x_i$