Shortest (least weight) paths with BFS tree and edge weights

• Bellman-Ford algorithm
• Each node $p$ has a variable $dist$ representing distance to root. Initially $p.dist = \infty$, $\text{root}.dist = 0$
• In each round, each node sends its $dist$ to all neighbors
• If for neighbor $q$ of $p$: $q.dist + w(p,q) < p.dist$
  – Then set $p.dist = q.dist + w(p,q)$
Shortest (least weight) paths with BFS tree and edge weights

• **Complexity**
  – (when all edge weights are positive)
  – Time: $n-1 \approx O(n)$
  – Message: $O(n^* |E|)$

• Also works for directed graphs
Weighed diameter

• In a weighted graph, the weighted diameter or weight-diameter is the
• Largest weight of the least weight path between 2 nodes
Bit complexity of communication

- We have assumed that each communication is 1 message, and we counted the messages.
- Sometimes, communication is evaluated by bit complexity – the number of bits communicated.
- This is different from message complexity because a message may have number of bits that depend on \( n \) or \( |E| \) or Diameter.

- For example, A routing table may be sent in a message, and a routing table has size \( O(n) \).

- In practice, data of size \( O(\log n) \) can be assumed to fit in a single message. E.g. node id.
- Data of size polynomial of \( n \): \( O(n), O(\sqrt{n}) \) etc need corresponding message sizes.
Distributed Systems

Systems and models

Rik Sarkar
James Cheney

University of Edinburgh
Spring 2014
Models

• Assumptions we make about the system
• Necessary to reason about systems

• Real world is too elaborate, too detailed

• We must discard unnecessary details and focus on the essentials
• Sometimes we may not know details in advance, when designing the system.
  – Our design must be general enough that they do not depend on these details
Models

• No one right way to model
• Always depends on the system and application in question
• Very often we do not know exactly where our design will be used
  – Try to make worst case assumptions that still give reasonable performance
• Today we discuss some elements of distributed systems that must be modeled, and some common aspects to keep in mind
Things to model

- Hardware
- Energy
- Communication
- Architecture: How software components are related
- Failures
- Computation
- Time and synchronization
- Security
- Mobility
Hardware

• Heterogeneity: Different nodes may have different properties
  – Speed of CPU
  – Memory
  – Storage
  – Polynomial of n memory/storage can be problematic
  – We can:
    • Try to model a few different types of nodes, specially when we know exactly which nodes will be of what type. E.g. Hand built cluster for a specific purpose
    • Or we can assume all nodes to be low power. E.g. sensor networks
  – In general, try to keep computation, memory and storage requirement *per node* as low as possible
Energy

• Important to prevent heating and to save battery
• Computation and communication cost energy
• In data centers processing “big data”
  – Keeping consumption low is critical
  – To keep down energy costs
  – To keep heating under control
• Google, Facebook spend millions on:
  – Cooling
  – Airflow
  – Power distribution
  – Measuring
  – modeling
  – Building data centers in the Arctic..
Energy

• In mobile/sensor devices
  – Energy is stored in battery
  – Consumption must be low to save to battery
• Design systems/algorithms to use less energy
• Understand and model energy usage to design better systems/algorithms
  – E.g. Energy consumption in wireless communication has complex properties. Depends on distance, interference, remaining battery etc..
Communication

• Each *process* may be in a different machine, and require network to send message to others

• Processes may be on the same computer (different programs, or threads) and communicate through shared memory.
  – Faster and less costly communication
Communication

Communication model is possibly the most important step affecting distributed design

- Broadcast (all nodes hear each message)
- Point to point communication between each pair of nodes (complete graph)
- Network as a general graph
- Communication through shared memory
  - For nodes on the same machine
Communication

• Network as a graph can be used to represent both shared memory and message based communication
  – E.g. we can put lower weights on shared memory communication
  – What are reasonable weights?
  – Are negative weights permissible?
• Shared memory can be simulated
  – For example everyone can have a copy of the memory, that has to be updated on each event
  – Not very efficient since $n$ updates must be made each time
Communication

• Broadcast not represented by a graph
• We can draw a complete graph
  – But this does not say that one transmission will reach all neighbors
• In practice, broadcast medium is still usually used for point to point communication
• So a graph is still a good representation
Point to point communication

- A sends a message to B
- How does A know that B received it?
- B sends an acknowledgement
- If A does not receive ack, A retransmits
The drawback of broadcast

- A sends a message to all neighbors
- A does not know if all neighbors received it
- What if all neighbors send acks?
  - That costs $n$ messages and time
  - Defeats the point of broadcast
Broadcast

• Good for cases where individual messages are not critical e.g. streaming video
• Bad for important messages
Communication: Overlay network

- We may sometimes ignore parts of the network
  - Nodes that carry messages but do not directly participate
  - Or edges that exist but we are not using
- Often used in peer-to-peer networks
Communication: Overlay network

- E.g. We may ignore routers, we may ignore edges that do not directly participate
- We may include edges that do not exist in reality, but are used in communication
- Depends on application
- The overlay may have no similarity to the physical network
Communication

• Remote procedure calls
  – Process A calls a function $f$ in the code of process B
    – This is equivalent to A sending a specific type to message to B, on reading which B decides to run the function $f$
  – RPC is a programming abstraction that makes some types of code easier
  – Does not change our fundamental concepts of a distributed systems
Architectures

• Layered software:
  – Different layers deal with different things
  – Well defined tasks for layers, upper layer assumes lower layer is doing its job
  – E.g. network protocols
Architectures

• Client – server
  – Servers do the computation
  – Clients request computations
Architecture

• Peer to peer
  – All nodes are equivalent (equal capabilities)
  – Each can (does) as client as well as server
  – May not be clear distinction between who is requesting and who is performing tasks
  – More general than client-server
Failures

• Nodes may fail
  – Hardware failure
  – Run out of energy or power failure
  – Software failure (crash)
  – Permanent
  – Temporary (what happens when it restarts? Recovers the state? Starts from initial state?)
  – Model depends on system. E.g. different types of failures occur with corresponding probabilities
Node failures

• Common abstract models
  – Stopping failure: node just stops working
    • May need assumptions about which computation/communication it finishes before stopping
    • May need assumption about neighbors knowing of failure
  – Byzantine failure: node behaves as an adversary
    • Imagine your enemy has taken control of the node
    • Is trying to spoil your computation

• Nodes may fail individually
  – E.g. each node fails with probability p

• Nodes may have correlated failure
  – E.g. all nodes fail in a region (data center, sensor field)
Link/communication failure

- May be temporary/permanent
- May happen due to
  - Hardware failure
  - Noise: electronic devices (microwaves etc) may transmit radio waves at similar frequencies and disrupt communication
  - Interference: Other communicating nodes nearby may disrupt communication
- Effects
  - Channel silent and unusable (hardware failure)
  - Channel active, but unusable due to noise and interference
  - Channel active, but may contain erroneous message (may be detected by error correcting codes)
Computation

• Synchronous:
  – Operation in rounds
  – In a round, a node performs some computation, and then sends some messages
  – All messages sent at the end of round $x$ are available to recipients at start of round $x+1$
    • But not earlier
Communication

• Synchronous
  – Can be implemented if message transmission time is bounded by some constant say m
  – Computation times for all nodes are bounded by some constant c
  – Clocks are synchronized
  – Then set each round to be m+c in duration
Asynchronous Communication

• No synchronization or rounds
  – Nodes compute at different and arbitrary speeds
  – Messages proceed at different speeds: may be arbitrarily delayed, may be received at any time

• Worst case model
  – No assumption about speeds of processes or channel
  – (But does not include communication/computation errors)
Asynchronous Communication

• Harder to manage
  – Message can arrive at any time after being sent, must be handled suitably
  – Possible to make some simplifying assumptions
    E.g.:
    • Channels are FIFO: order of messages on a channel are preserved
    • Some code blocks are atomic (not interrupted by messages)
    • Either communication or computation times bounded
Synchronous communication in Real systems

- Synchronous communication can be a fair model
- Modern computers and networks are fast
  - (though not arbitrarily fast)
- Easier to design algorithms and analyze
- Well designed algorithms are faster and more efficient
- Often can be adapted to asynchronous systems
  - Often a starting point for design
Security

• Issues:
  – Unauthorized access, modification. Making systems unavailable (DOS)
  – Attack on one or more nodes
    • Causing to it fail
    • Read data
    • Taking control to read future data, disrupt operation
  – Attack on communication links/channel
    • Block communication
    • Read data in the channel (easy in wireless without encryption)
    • Corrupt data in the channel
Security

• Solutions usually have specific assumptions of what the adversary can do

• E.g. If adversary has access to channel
  – Cryptography may be able to prevent reading/corrupting data
Mobility

• Movement makes it harder to design distributed systems
  – Communication is difficult
    • Delays, lost messages
    • Edge weights can change
  – Applications that depend on location must adapt to movement
• How do people move? What is a model of movement?
  – Not yet well understood
Modeling distributed systems

• Many possibilities
• Choose your assumptions carefully for your problem
• Pay close attention to what is known about communication/network
• Start with simpler models
  – Usually more assumptions, fewer parameters
  – See what can be achieved
  – Then try to drop/relax assumptions