Distributed Algorithms in Networks
EECS 122: Lecture 17

Department of Electrical Engineering and Computer Sciences
University of California
Berkeley

March 21, 2006

The Internet is a HUGE Distributed System

- Nodes are local processors
- Messages are exchanged over various kinds of links
- Nodes contain sensors which sense local changes
- Nodes control the network jointly
  - Method for doing this is a distributed algorithm
  - Example: Routing
- Time taken to solve the problem has two components:
  - Computation time taken for local processing
  - Communication time for messages to be received over the links

Solving Global Problems in a Distributed Setting

- Examples:
  - Minimum Spanning Tree
  - Shortest Path
  - Leader Election
  - Topology Broadcast
- Much easier to think in terms of centralized algorithms
  - Creativity needed to convert to the distributed case

Today

- Focus on protocol design issues
- How to move from Centralized to Distributed Alg.
- Synchronous and Asynchronous computation
  - Why does the Asynchronous Bellman Ford converge?
- Selfish behavior distributed systems

The Network is Heterogeneous

- Speed
  - Dialup to terabit fiber
- Reliability
  - Hosts: Distributed Server farms to 486 PC
  - Links: Noisy wireless to virtually error free fiber
- Congestion
- Trustworthiness
- What is a general enough model to cover all of this?

Network Protocols often have unintended effects

- TCP
  - Example 1
    - TCP connections detect congestion after it has happened
    - May cause packet drops from un congested “well behaved flows”
    - Non congested flows back off
  - Example 2
    - Two TCP flows sharing the same router get uneven bandwidths
    - because one has a much smaller RTT than the other
    - Routing
    - Oscillation and countless other pathologies
    - It is very difficult to avoid these unintended effects
**Consensus over an Unreliable Link**

- A and B in a connection over an unreliable link.
- They both want to terminate the connection only if they are certain that no more packets will arrive from the other user.

- A won’t terminate unless it knows that B knows it is about to terminate.
- B won’t terminate unless it knows that A knows it is about to terminate.

**Consensus Problem**

- Suppose B tells A it can terminate and A receives this message, say M.
- A can terminate, but B will never know if A actually received M and so it can’t terminate.

- A sends ACK(M) to B, but then A needs to make sure that B received this message, so it must wait for ACK(ACK(M))...
- A never terminates.
- In fact, NO protocol exists to solve this problem!
- Worth convincing yourself of this fact.

**Link model**

- Error correction
  - Assume that errors can “eventually” corrected
- Propagation Delay
  - Fixed
  - Variable but no more than d
  - Variable with no upper bound
- Other components of delay
  - Queueing Delay
  - Transmission Delay
- Packet order
  - FIFO
  - Can be delivered in arbitrary order

**Maintaining accurate topology information**

Whenever a link goes down/up, its end points send messages to all their neighbors who then flood.

1. CD fails
Maintaining accurate topology information

Whenever a link goes down/up, its end points send messages to all their neighbors who then flood.

1. CD fails
   - A marks the link down
2. CD comes back up

This can be fixed with sequence numbers, but then other problems emerge…

Synchronous v/s Asynchronous Algorithms

- Synchronous algorithms can be described in terms of global iterations. The time taken for a given iteration is the time taken for the slowest processor to complete that iteration: *time driven*
  - E.g. TDM or SONET
- Asynchronous algorithms execute at a processor based on received messages and internal state: *event driven*
  - E.g. IP protocols which must run over heterogeneous systems

Slotted Time

- Slotted system 1, 2, ..., 3…
  - All nodes agree on slot boundaries
    - “Have access to a global clock”
  - Helps to co-ordinate the nodes
    - Every node can run the same algorithm
      - Proving correctness is generally tractable if the centralized algorithm is analyzable
      - Easier to understand the sequence of communication between nodes
Synchronous Bellman-Ford (SBF)

- Every node runs the same algorithm
- Time is slotted and in every tick each node sends its distance vector.
- At time $h$, node $i$ has as an estimate of the shortest path to node 1 that has $\leq h+1$ hops
  $$D_{h+1}(i,j) = \min_{k \in N(i)} \{D_h(k,j) + c(i,k)\}$$

Synchronous Scheduling

- Great when links are reliable and similar...
- But what when some links are much faster?
  - Node 5 suffers synchronization penalty

Synchronous Timing

- Slow nodes can create a penalty as well
- Penalty can be huge!

Implementing a Synchronous Algorithm

- Suppose the slowest process can complete an iteration in time $T_p$
- Link delay is always less than $T_l$
- Then a slot size of $T_p + T_l$ or more is sufficient
  - But most processors may be idle most of the time
  - What if $T_p$ and or $T_l$ are not known?

Locally Synchronous Computation

- Forget about fixed slots
- When a node has received all round $k-1$ messages from its neighbors, it computes and sends out its round $k$ message
- Worst-case: As slow as synchronous computation
- Generally much faster
- Any synchronous algorithm that isn’t using time as a part of the computation will also work when run in a locally synchronous manner.
Local Synchronization

Send update $k$ after you’ve heard update $k-1$ from all neighbors.

Node 3

Node 4

Node 5

Why bother with Asynchronous Algorithms

- To reduce the synchronization penalty
- Difficult to get the synchronous algorithm to start
- The network is dynamic
  - Flows
  - Topology
    - Think of the algorithm having to “restart” with a new set of initial conditions, every time there is a failure
- Changes create “events” which may or may not have global impact
  - Event-driven algorithms better suited

Asynchronous Bellman Ford (ABF)

- Don’t even wait to hear from all neighbors!
  - Use most recent information to compute new distance vectors
    - i.e. use last received values of $D(i)$ and $d$
  - Whenever ready, each node $i$ computes
    - $D(i) = \min_{k \in N(i)} [D(k) + c(i,k)]$
    - Sends the result to each of its neighbors
  - No notion of global iterations
- In general, nodes are using different and possibly inconsistent estimates

Asynchronous computation

No notion of “slot size” at all

Why should this work?

Asynchronous Bellman Ford

- Regardless of how asynchronous the nodes are, the algorithm will eventually converge to the shortest path
  - Links can go down and come up – but as long as the topology is fixed after some time $t$, the algorithm will eventually converge to the shortest path
  - Why?
    - There’s some hope because the $D(j)$ can only go up if one of $j$’s neighbors estimates has gone up.
Idea

- There are too many different “runs” of ABF, so let’s try to bound the range of distance estimates of D(j) over time.
- Do this by two different runs of Synchronous BF
  - Set different initial estimates
    - One run U uses the familiar ones, i.e., estimate is infinity if no edge
    - The other, L, uses -1 if no edge!
  - One bounds the estimates from above, one from below and both find the correct shortest paths eventually.
- For every iteration k of the two SBF runs
  - \( L_k(j) \leq L_{k+1}(j) \leq D^*(j) \leq U_{k+1}(j) \leq U_k(j) \)
- For any asynchronous run, A, it is possible to show that for any k, there is a time t such that
  - \( L_k(j) \leq L_{k+1}(j) \leq A_t(j) \leq U_{k+1}(j) \leq U_k(j) \)
- Since both lower and upper runs converge to the optimal, so will ABF eventually.

Soft State

- State with Time-Out
- Example: A host joins a group by sending a “join” message to a “host manager”. The manager adds the host to the group for the next T seconds. If the host wants to stay in the group it must send a refresh message within T seconds to the manager. Otherwise it is dropped.
- Advantage: Manager robust to host failure
- Disadvantage: Too many messages
- Most internet protocols use this way of communicating
- Trades of simplicity of correctness with complexity of communication

Trustworthiness

- Three levels
  - Honest: Always in conformance of the protocol
  - Selfish: May try to get better performance out of the protocol (BGP)
  - Malicious: Unpredictable
- Internet Protocols (for the most part) assume Honest protocol agents
- Unreliable infrastructure
- Infrastructure has gotten more reliable, and agents have gotten less honest...
- Braess’s Paradox: Example of how Greediness and distributed algorithms can lead to suboptimality

Congestion Sensitive Routing

- Each Node is Greedy
- Weights are delays/bit
  - 1 unit of traffic from s to t
  - u bits on the upper path
  - 1-u bits on the lower path
- Depends on traffic
  - Node S minimizes
    - Total delay = \( u(u+1) + (1-u)(2-u) = 2u^2 - u + 1 \)
    - Delay minimized at \( u = 0.5 \)
    - So Total Delay = 1.5 s

The nature of asynchronous distributed protocols

- Generally non-intuitive
- Limited theory to work with
  - Correctness extremely hard to prove
  - Robustness hard to analyze
- Networking gurus have a vast knowledge of special cases that can lead to strange behaviors
  - Misconfiguration is a big cause of errors
- Soft state helps a lot, but wastes many messages!
- What about just broadcasting topology information accurately so that these problems go away…
Greediness leads to suboptimality

S still sends .5 on each path

5

R is greedy
R diverts all .5 units on to the new link

Weights are delays/bit
1 unit of traffic from s to t
u bits on the upper path
1-u bits on the lower path

BRAESS’S PARADOX

Now total delay is 2!

Conclusions

- Distributed Algorithms are not intuitive
- There is no systematic way to design them
  - Active research area is making some progress
  - Until then use
    - Hacking Abilities
    - Simulation
    - Control Theory
    - Optimization
    - Graph Theory
    - Game Theory
    - …
- Greedy and malicious users complicate the protocol design problem even more
  - Another active research area making progress
- This is why it is hard to build networks…