CS 336 Lecture 23– 4/15/98

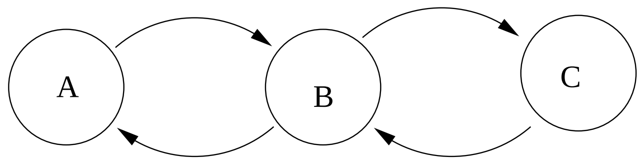
Announcements

- Project designs due Friday
- Self-scheduled midterm available from my office or dept. secretary now until Friday.
- I have put a copy of the midterm from my last offering of this course (when it was a tutorial) online.
- I have updated the syllabus with some readings on the Internet. They will extend to our discussion of several topics over the next two weeks.
- Questions on the programming assignment? Which language to use?

Dynamic Behavior of Shortest Path Routing Algorithms

1. Last time, we considered the degree to which the Bellman-Ford algorithm and Dijkstra's shortest path algorithm seemed appropriate for implementation in a distributed environment.
   - The nature of the Bellman-Ford algorithm suggests a simple distributed implementation in which all necessary communications occurs between directly connected network nodes.
   - By contrast, Dijkstra's algorithm appears to require far more extensive communications if used in a distributed setting. Basically, each node needs to receive from every other node periodic messages describing the state of each attached outgoing link.
2. There is, however, another important criterion for judging a routing algorithm — its ability to adjust to changes in the topology of or load on the network. We now want to consider this issue.
3. Basically, Dijkstra's algorithm will clearly adapt rapidly if we solve the communication problems.
   - If we can disseminate information on changes in the delays associated with links in a timely manner, then since the algorithm starts from scratch each time we run it, routes will adjust as soon as the algorithm is rerun.
4. The Bellman-Ford algorithm will also adapt (note the absence of the term "rapidly").
   - While I have not and will not provide a proof of my claim, last time I stated a very strong claim for the distributed version of the Bellman-Ford algorithm. Namely, the algorithm will converge to the correct distance vector values at all nodes in a finite amount of time regardless of the initial estimates of the distances used.

     So, if the algorithm runs until correct distances are computed and then some link costs change, letting the algorithm just continue to run is equivalent to starting it with the old distance values as initial estimates. So, it will eventually re-converge to the correct distances associated with the new link costs.
5. Just in case you couldn't see it coming, the problem with the distributed Bellman-Ford algorithm is that it may not converge to correct values after a link cost changes in a timely manner.
   - To understand the problem, consider the simple network topology shown below:



   - Assume that initially all link costs are 1. In this case, after a while the Bellman-Ford algorithm will determine that the path costs between the nodes in the graph have the values shown in the following table:

	A	B	C
A	0	1	2
B	1	0	1
C	2	1	0

   - Now, assume that B begins sending a great deal of traffic to C filling the queue from B to C so that the cost associated with $d_{B\rightarrow C}$ becomes 100. Suppose, that at this point, B receives a packet from A containing A's distance vector. B will recompute its own distance vector using A's information and its own current link costs.
     - When B recomputes $D_{B\rightarrow C}$, it will have to consider paths to C starting with a step to each of its neighbors, A and C. Based on the information available to B, the costs of these paths will be:
       * Starting with C = $d_{B\rightarrow C} + D_{C\rightarrow C} = 100 + 0$
       * Starting with A = $d_{B\rightarrow A} + D_{A\rightarrow C} = 1 + 2$

– So, B will conclude that its best distance to C is 3 and that it is associated with the path from B to C that starts by going to A! This yields the new "global" state:

	A	B	C
A	0	1	2
B	1	0	3
C	2	1	0

- Having updated its own distance vector, B will now feel obligated to forward its latest information to its neighbors. So, A will shortly receive B's new D values and recompute its figures. A will still think the best way to reach C is through B, but since B's distance vector advertises a higher cost to reach C, A's will have to adjust its distance to C yielding:

	A	B	C
A	0	1	4
B	1	0	3
C	2	1	0

- Hopefully, you can see that next B will decide that while going through A to get to C still looks like the best approach the cost of the path must be increased to 5. A will then update its cost to C to 6. Basically, at each step either A or B will increase its estimate of the cost to C by 1 until finally B realizes that it is actually cheaper to go direct to C.
- Based on the worst case scenario (a link actually goes down making its cost infinite), this behavior of the distributed Bellman-Ford algorithm is know as the "count to infinity" problem.

6. This strange behavior of the Bellman-Ford algorithm is obviously undesirable:

   - While A and B are confused about their routing tables, they will both treat packets destined for C very badly. A will send them to B which will send them to A which will send them back to B, ...

     This is known as a routing loop.
   - Eventually, the sender of a node caught in a routing loop will time out and retransmit the packet producing another node caught in a loop.
   - If enough packets get caught in the loop, they will increase the congestion on the links and delay the many distance vector updates required to eventually resolve the confusion.
   - The increased congestion due to loops can lead to large increases in link costs which triggering other attempts to update D values which may lead to more loops.

7. There are ways to combat the silliest examples of this behavior.

   - A variant of the distributed Bellman-Ford algorithm called "split-horizon" attempts to fix the count to infinity problem by having nodes lie a little when reporting their distance vectors to neighbors. In particular, under "split horizon" if node i thinks the best path to j begins with k then it sends infinity as its estimate of $D_{i \to j}$ to k.

   However, more complex examples of slow convergence are still possible on cycles involving more than two nodes.

8. As a result, Dijkstra's shortest path algorithm is considered the better choice for route computation in networks based on shortest path routing.

   This means we have to consider the problem of how to distribute the link-state information it requires.