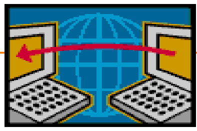


Even More Routing

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson and other colleagues at Princeton and UC Berkeley

Questions about Project 1

- Colin goes into cone of silence for next 30 hours
- So ask your questions now!

Today's Lecture: A little of everything

- Finishing up distance vector routing
  - Last time we covered the ***good***
  - This time we cover the ***bad*** and the ***ugly***
- Covering some "missing pieces"
  - Maybe networking isn't as simple as I said….
- Lots of details today…
  - So I will go slowly and ask you to do the computations
  - Will have you ask your neighbors if you can't figure it out
    - If they can't figure it out, sit next to smarter people next time!

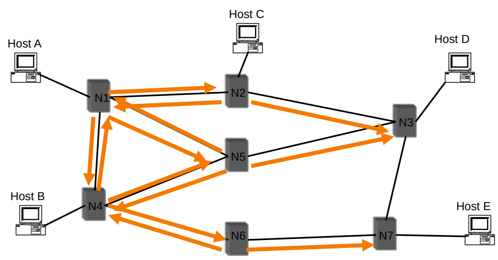
Two Ways to Avoid Loops

- **Global state, local computation**
  - Link-state
  - Broadcast local information, construct network map
- **Local state, global computation**
  - Distance-Vector
  - Minimizing "cost" will produce loop-free routes
  - Iterative computation: no one knows the topology

Distance Vector Routing

- Each router knows the links to its neighbors
  - Does *not* flood this information to the whole network
- Each router has provisional "shortest path"
  - E.g.: Router A: "I can get to router B with cost 11"
- Routers exchange this *Distance-Vector* information with their neighboring routers
  - Vector because one entry per destination
  - *Why only advertise "best" path? Why not two best?*
    - *Loops and lies….*
- Routers look over the set of options offered by their neighbors and select the best one
- Iterative process converges to set of shortest paths

Information Flow in Distance Vector



Bellman-Ford Algorithm

- INPUT:
  - Link costs to each neighbor
  - Not full topology
- OUTPUT:
  - Next hop to each destination and the corresponding cost
  - Does not give the complete path to the destination
- My neighbors tell me how far they are from dest'n
  - Compute: (cost to nhbr) plus (nbr's cost to destination)
  - Pick minimum as my choice
  - Advertise that cost to my neighbors

7

Bellman-Ford Overview

- Each router maintains a table
  - Best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$
- Each local iteration caused by:
  - Local link cost change
  - Message from neighbor
- Notify neighbors *only* if least cost path to any destination changes
  - Neighbors then notify their neighbors if necessary

Each node:

wait for (change in local link cost or msg from neighbor)

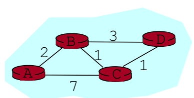
recompute distance table

if least cost path to any dest has changed, *notify* neighbors

8

Bellman-Ford Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ⇒ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$



Node A

	B	C
B	2	8
C	3	7
D	4	8

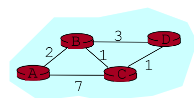
Neighbor (next-hop)

Destinations

$D_C(A, D)$

Bellman-Ford Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X ⇒ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$



Node A

	B	C
B	2	8
C	3	7
D	4	8

Smallest distance in row Y = shortest Distance of A to Y, D(A, Y)

Distance Vector Algorithm (cont'd)

```
 Initialization:
  for all neighbors V do
     if V adjacent to A
        D(A, V) = c(A,V);
     else
        D(A, V) = ∞;
   send D(A, Y) to all neighbors
loop:
  wait (until A sees a link cost change to neighbor V  /* case 1 */
       or until A receives update from neighbor V)   /* case 2 */
  if (c(A,V) changes by ±d)  /* ⇐ case 1 */
       for all destinations Y that go through V do
          D_V(A,Y) = D_V(A,Y) ± d
  else if (update D(V, Y) received from V) /* ⇐ case 2 */
       /* shortest path from V to some Y has changed  */
       D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
  if (there is a new minimum for destination Y)
       send D(A, Y) to all neighbors
  forever
```

- **c(i,j): link cost from node *i* to *j***
- **$D_Z(A,V)$: cost from A to V via Z**
- **D(A,V): cost of A's best path to V**

11

Distance Vector Algorithm (cont'd)

Each node: initialize, then

wait for (change in local link cost or msg from neighbor)

recompute distance table

if least cost path to any dest has changed, *notify* neighbors

12

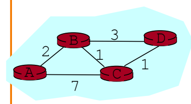
Distance Vector Algorithm (cont’d)

```
Initialization:
  for all neighbors V  do
      if V adjacent to A
          D(A, V) = c(A,V);
      else
          D(A, V) = ∞;
    send D(A, Y) to all neighbors
loop:
  wait (until A sees a link cost change to neighbor V  /* case 1 */
          or until A receives update from neighbor V)    /* case 2 */
   if (c(A,V) changes by ±d)  /* ⇐ case 1 */
          for all destinations Y that go through V do
               D_V(A,Y) =  D_V(A,Y) ± d
   else if (update D(V, Y) received from V) /* ⇐ case 2 */
          /* shortest path from V to some Y has changed  */
          D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
   if (there is a new minimum for destination Y)
          send D(A, Y) to all neighbors
  forever
```

- **c(i,j):** link cost from node *i* to *j*
- **$D_Z(A,V)$:** cost from A to V via Z
- **D(A,V):** cost of A’s best path to V

13

Example: Initialization



Node A

	B	C
B	2	∞
C	∞	7
D	∞	∞

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

```
Initialization:
for all neighbors V  do
    if V adjacent to A
        D(A, V) = c(A,V);
    else
        D(A, V) = ∞;
  send D(A, Y) to all neighbors
```

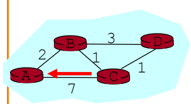
Node C

	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

Example: C sends update to A



Node A

	B	C
B	2	8
C	∞	7
D	∞	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

$D_C(A, B) = D_C(A,C) + D(C, B) = 7 + 1 = 8$

$D_C(A, D) = D_C(A,C) + D(C, D) = 7 + 1 = 8$

```
loop:
…
else if (update D(A, Y) from C)
  D_C(A,Y) = D_C(A,C) + D(C, Y);
if (new min. for destination Y)
  send D(A, Y) to all neighbors
forever
```

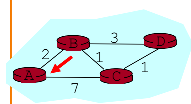
Node C

	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

Example: Now B sends update to A



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

…C) = $D_B(A,B) + D(B, C) = 2 + 1 = 3$

…(A,B) + D(B, D) = 2 + 3 = 5

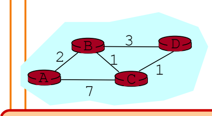
Make sure you know why this is 5, not 4!

```
else if (update D(A, Y) from B)
  D_B(A,Y) = D_B(A,B) + D(B, Y);
if (new min. for destination Y)
  send D(A, Y) to all neighbors
forever
```

	[illegible]	[illegible]	[illegible]
A	7	∞	∞
B	∞	1	
D	∞	∞	1

	[illegible]	C
A	∞	∞
B	3	∞
C	∞	1

Example: After 1st Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	8	∞
C	9	1	4
D	∞	2	3

Make sure you know why this is 3

Assume all send messages at same time

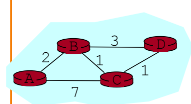
	[illegible]	B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

Example: No…

How could we fix this?



Node A

	B	C
B	2	8
C	3	7
D	5	8

	[illegible]	C	D
A	2	8	∞
C	5	1	4
D	7	2	3

$D_A(B, C) = D_A(B,A) + D(A, C) = 2 + 3 = 5$

$D_A(B, D) = D_A(B,A) + D(A, D) = 2 + 5 = 7$

```
loop:
…
else if (update D(B, Y) from A)
  D_A(B,Y) = D_A(B,A) + D(A, Y);
if (new min. for destination Y)
  send D(B, Y) to all neighbors
forever
```

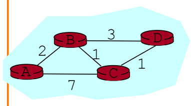
Node C

	A	B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

Example: End of 2nd Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	8
C	5	1	4
D	7	2	3

Node C

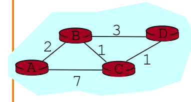
	A	B	D
A	7	3	6
B	9	1	3
D	12	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

Assume all send messages at same time

Example: End of 3rd Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	7
C	5	1	4
D	6	2	3

Node C

	A	B	D
A	7	3	5
B	9	1	3
D	11	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

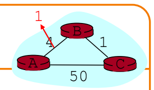
If you can't figure it out after three minutes, ask your neighbor

What route does this 11 represent?

Intuition

- Initial state: best one-hop paths
- One simultaneous round: best two-hop paths
- Two simultaneous rounds: best three-hop paths
- **The key here is that the starting point is not the initialization, but some other set of entries. Convergence could be different!**
- Must eventually converge
  - as soon as it reaches longest best path
- …..but how does it respond to changes in cost?

DV: Link Cost Changes



Node A

	Stable state: B	C	A-B changed: B	C	A sends tables to B, C: B	C	B sends tables to C: B	C	C sends tables to B: B	C
B	4	51	1	51	1	51	1	51	1	51
C	5	50	2	50	2	50	2	50	2	50

Node B

	Stable state: A	C	A-B changed: A	C	A sends tables to B, C: A	C	B sends tables to C: A	C	C sends tables to B: A	C
A	4	6	1	6	1	6	1	6	1	3
C	9	1	6	1	3	1	3	1	3	1

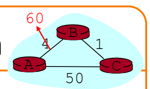
Node C

	Stable state: A	B	A-B changed: A	B	A sends tables to B, C: A	B	B sends tables to C: A	B	C sends tables to B: A	B
A	50	5	50	5	50	5	50	2	50	2
B	54	1	54	1	51	1	51	1	51	1

Link cost changes here

"good news travels fast"

DV: *Count to Infinity* Problem



Node A

	Stable state: B	C	A-B changed: B	C	A sends tables to B, C: B	C	B sends tables to C: B	C	C sends tables to B: B	C
B	4	51	60	51	60	51	60	51	60	51
C	5	50	61	50	61	50	61	50	61	50

Node B

	Stable state: A	C	A-B changed: A	C	A sends tables to B, C: A	C	B sends tables to C: A	C	C sends tables to B: A	C
A	4	6	60	6	60	6	60	6	60	8
C	9	1	65	1	110	1	110	1	110	1

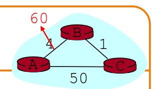
Node C

	Stable state: A	B	A-B changed: A	B	A sends tables to B, C: A	B	B sends tables to C: A	B	C sends tables to B: A	B
A	50	5	50	5	50	5	50	7	50	7
B	54	1	54	1	101	1	101	1	101	1

Link cost changes here

"bad news travels slowly" (not yet converged)

DV: *Poisoned Reverse*



- If B routes through C to get to A:
  - B tells C its (B's) distance to A is infinite (so C won't route to A via B)

Node A

	Stable state: B	C	A-B changed: B	C	A sends tables to B, C: B	C	B sends tables to C: B	C	C sends tables to B: B	C
B	4	51	60	51	60	51	60	51	60	51
C	5	50	61	50	61	50	61	50	61	50

Node B

	Stable state: A	C	A-B changed: A	C	A sends tables to B, C: A	C	B sends tables to C: A	C	C sends tables to B: A	C
A	4	∞	60	∞	60	∞	60	∞	60	51
C	∞	1	∞	1	110	1	110	1	110	1

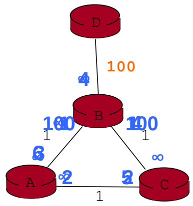
Node C

	Stable state: A	B	A-B changed: A	B	A sends tables to B, C: A	B	B sends tables to C: A	B	C sends tables to B: A	B
A	50	5	50	5	50	5	50	61	50	61
B	54	1	54	1	∞	1	∞	1	∞	1

Link cost changes here

Note: this converges after C receives another update from B

Will PR Solve C2I Problem Completely?



A few other inconvenient aspects

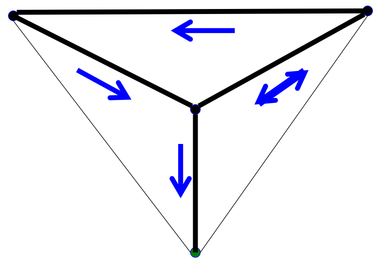
- What if we use a non-additive metric?
  - E.g., maximal capacity
- What if routers don't use the same metric?
  - I want low delay, you want low loss rate?
- What happens if nodes lie?

Can You Use Any Metric?

- We said that we can pick any metric. Really?
- What about maximizing capacity?

What Happens Here?

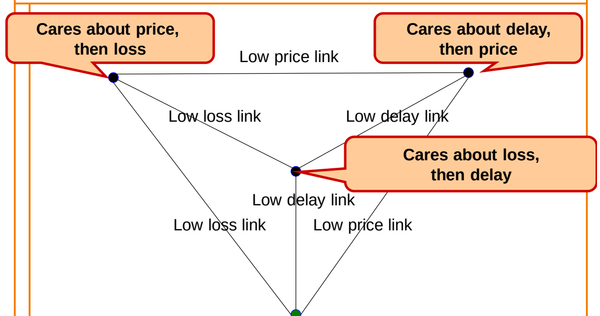
How could you fix this (without changing metric)?



No agreement on metrics?

- If the nodes choose their paths according to different criteria, then bad things might happen
- Example
  - Node A is minimizing latency
  - Node B is minimizing loss rate
  - Node C is minimizing price
- Any of those goals are fine, if globally adopted
  - Only a problem when nodes use different criteria
- Consider a routing algorithm where paths are described by delay, cost, loss

What Happens Here?



Would path-vector fix this?

Must agree on loop-avoiding metric

- When all nodes minimize same metric
- And that metric increases around loops
- Then process is guaranteed to converge

31

What happens when routers lie?

- What if router claims a 1-hop path to everywhere?
- All traffic from nearby routers gets sent there
- How can you tell if they are lying?
- Can this happen in real life?
  - It has, several times….

32

Routing: Just the Beginning

- Link state and distance-vector (and path vector) are the deployed routing paradigms
- But we know how to do much, much better…
- Stay tuned for a later lecture where we:
  - Reduce convergence time to zero
  - Deal with "policy oscillations"
  - Enable multipath routing

33