15-441

Computer Networking

Distance-Vector Routing
Sept. 29, 2004

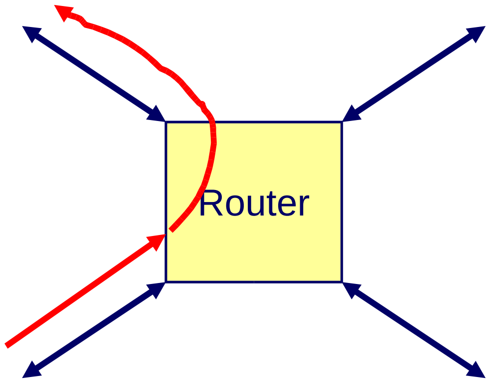
Topics

- Routing task
- Conceptual algorithm
- Realities
- RIP protocol

Slides

- Hui Zhang, Randy Bryant, Dave Eckhardt

Router Operation



When Packet Arrives at Router

- Examine header to determine intended destination
- *Look up in table to determine next hop in path*
- Send packet out appropriate port

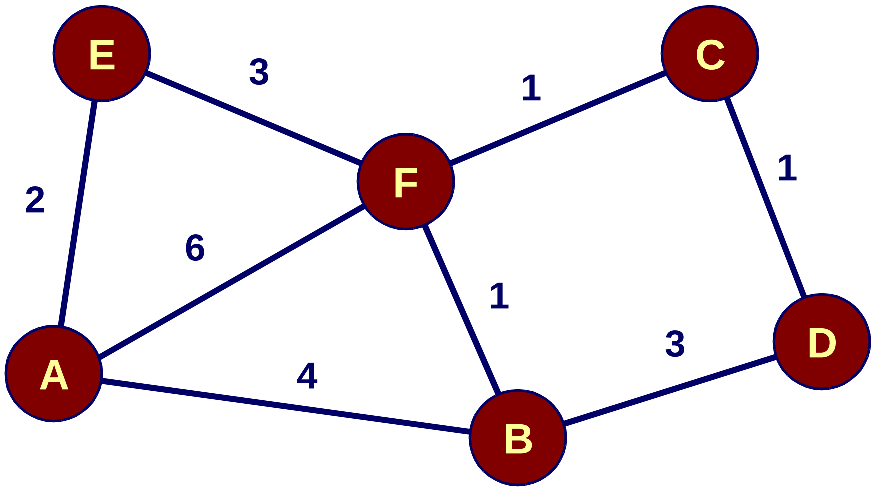
Terminology

- Each router *forwards* packet to next router
- Overall goal is to *route* packet from source to destination

Today's task

- How to generate the routing table

Graph Model



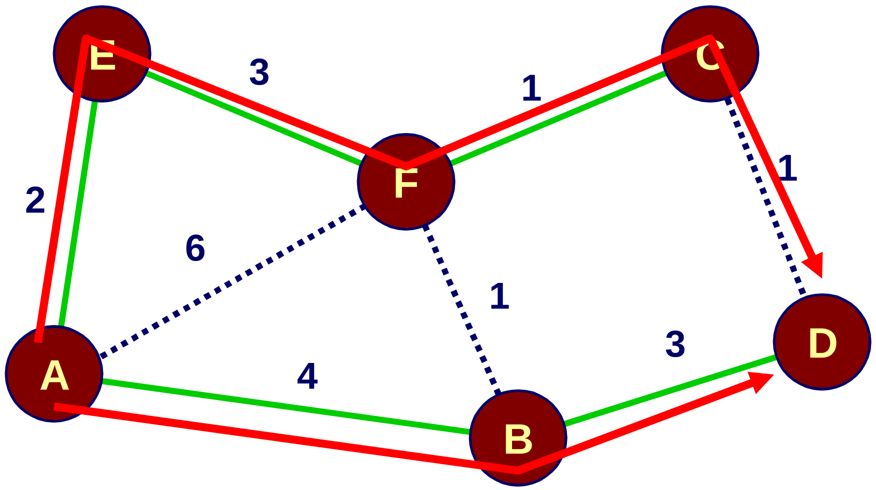
- Represent each router as node
- Direct link between routers represented by edge
  - Symmetric links ⇒ undirected graph
- Edge "cost" c(x,y) denotes measure of difficulty of using link

Task

- Determine least cost path from every node to every other node
  - Path cost d(x,y) = sum of link costs

Routes from Node A

Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	6	E
D	7	B
E	2	E
F	5	E

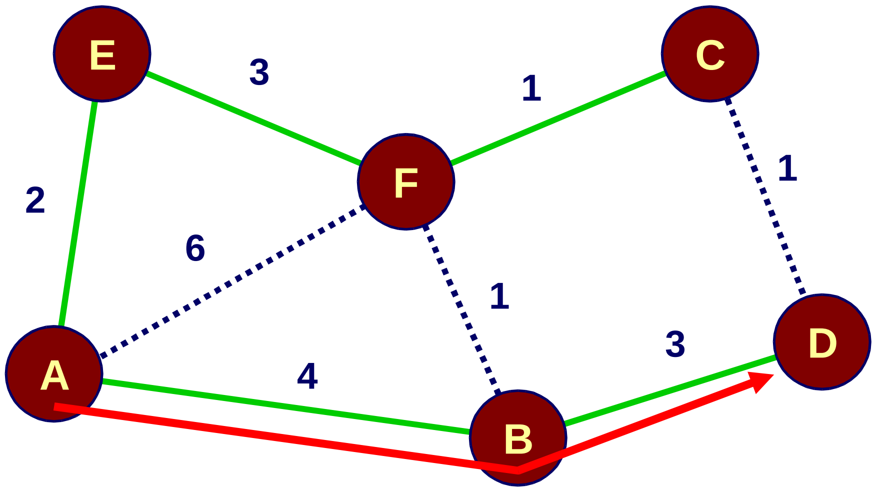


Properties

- Some set of shortest paths forms tree
  - (why is it a tree?)
  - "Shortest path spanning tree"
- Solution not unique
  - E.g., A-B-D, A-E-F-C-D both have cost 7

Will Packets Follow Computed Route?

Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	6	E
D	7	B
E	2	E
F	5	E



Intended Route

- A-B-D
- First hop B

Will Packets Follow Computed Routes?

Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	6	E
D	7	B
E	2	E
F	5	E

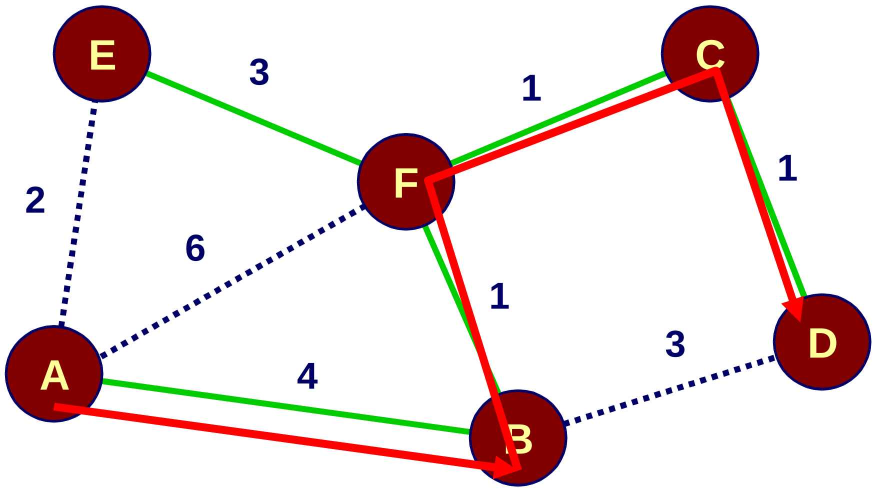


Table for B		
Dest	Cost	Next Hop
A	4	A
B	0	B
C	2	F
D	3	F
E	4	F
F	1	F

Intended Route

- A-B-D
- First hop B

Actual Route

- A-B-F-C-D
- B has different version of best path to D

Things to Think About

Given

- Each entry in each table specifies next hop along SOME shortest path

Concerns

- Could a packet get stuck in a loop?
  - What conditions would prevent this?
- Will a packet follow a shortest path?

Ways to Compute Shortest Paths

Centralized

- Collect graph structure in one place
- Use standard graph algorithm
- Disseminate routing tables

Partially Distributed

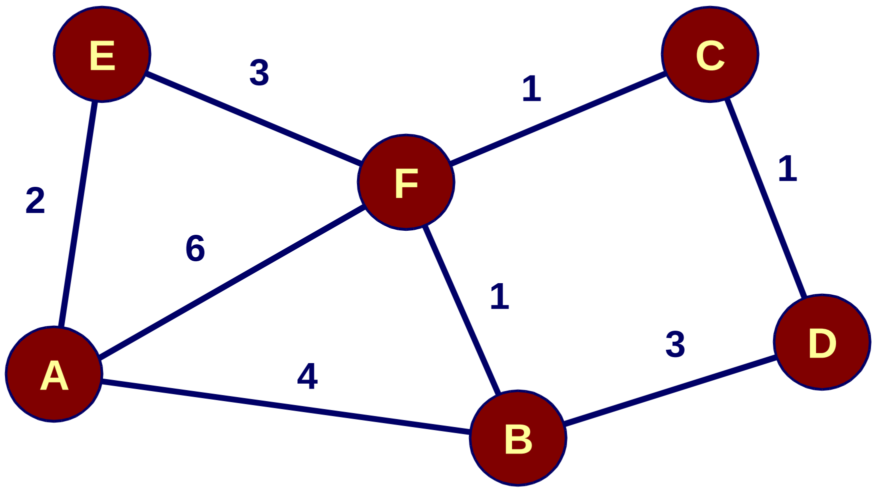
- Every node collects complete graph structure
- Each computes shortest paths from it
- Each generates own forwarding table
- "Link-state" algorithm

Fully Distributed

- No one has copy of graph
- Nodes construct their own tables iteratively
- Each sends information about ***<u>its table</u>*** (vs. graph) to neighbors
- "Distance-Vector" algorithm

Distance-Vector Method

Initial Table for A		
Dest	**Cost**	**Next Hop**
A	0	A
B	4	B
C	∞	–
D	∞	–
E	2	E
F	6	F



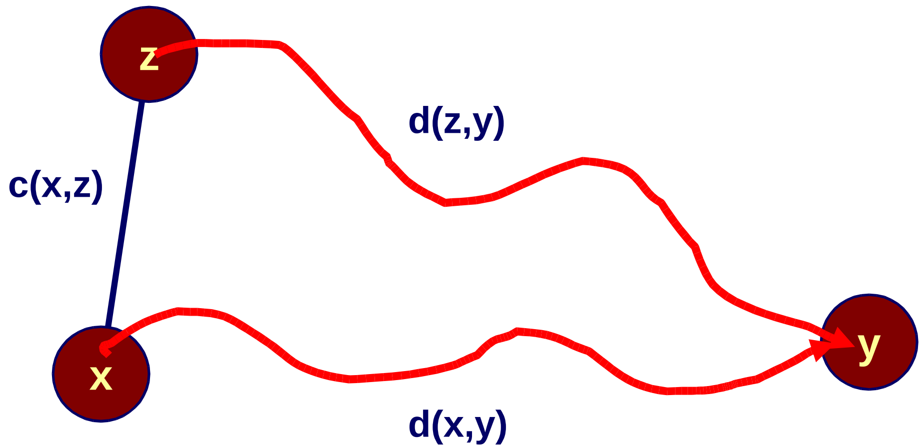
Idea

- At any time, have (cost,next-hop) of best known path to destination
- Use cost ∞ when no path known

Initially

- Have entries only for directly connected nodes

Distance-Vector Update



Update(router=x, dest=y, peer=z)

- $d \leftarrow c(x,z) + d(z,y)$ # Cost of path from x to y with first hop z
- if $d < d(x,y)$

  # Found better path

  return d,z # Updated cost / next hop
- else

  return d(x,y), nexthop(x,y) # Existing (cost, next hop)

Synchronous Version

- **Bellman-Ford algorithm**

Repeat

For every hop z

For every source x

For every destination y

d′(x,y) $\leftarrow$ Update(x,y,z)

For all x,y: d(x,y) $\leftarrow$ d′(x,y)

Until Converge

- **What is maximum number of iterations?**

Synchronous Start

Optimum 1-hop paths

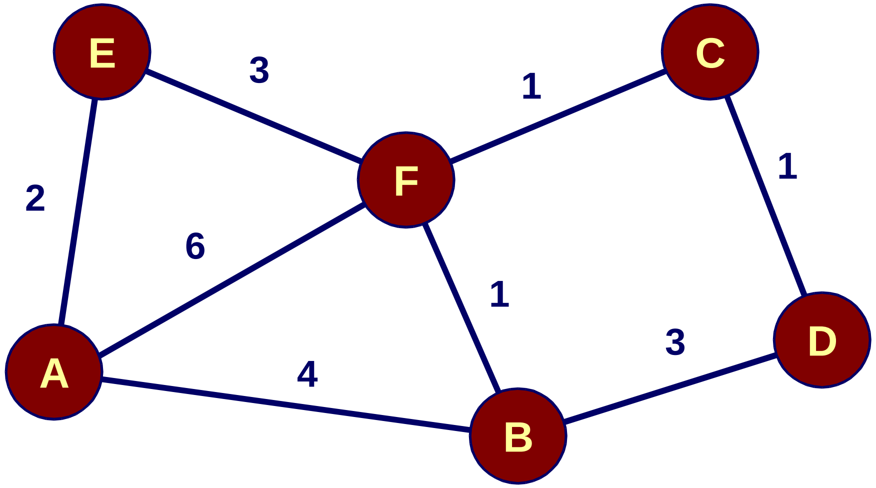


Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	∞	–	A	∞	–	A	2	A	A	6	A
B	4	B	B	0	B	B	∞	–	B	3	B	B	∞	–	B	1	B
C	∞	–	C	∞	–	C	0	C	C	1	C	C	∞	–	C	1	C
D	∞	–	D	3	D	D	1	D	D	0	D	D	∞	–	D	∞	–
E	2	E	E	∞	–	E	∞	–	E	∞	–	E	0	E	E	3	E
F	6	F	F	1	F	F	1	F	F	∞	–	F	3	F	F	0	F

Synchronous Iteration #1

Optimum 2-hop paths

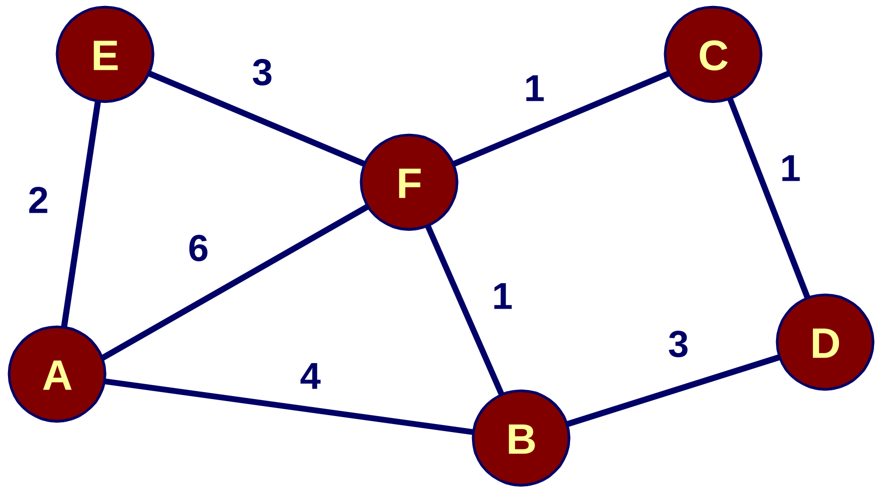


Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	7	F	A	7	B	A	2	A	A	5	B
B	4	B	B	0	B	B	2	F	B	3	B	B	4	F	B	1	B
C	7	F	C	2	F	C	0	C	C	1	C	C	4	F	C	1	C
D	7	B	D	3	D	D	1	D	D	0	D	D	∞	–	D	2	C
E	2	E	E	4	F	E	4	F	E	∞	–	E	0	E	E	3	E
F	5	E	F	1	F	F	1	F	F	2	C	F	3	F	F	0	F

Synchronous Iteration #2

Optimum 3-hop paths

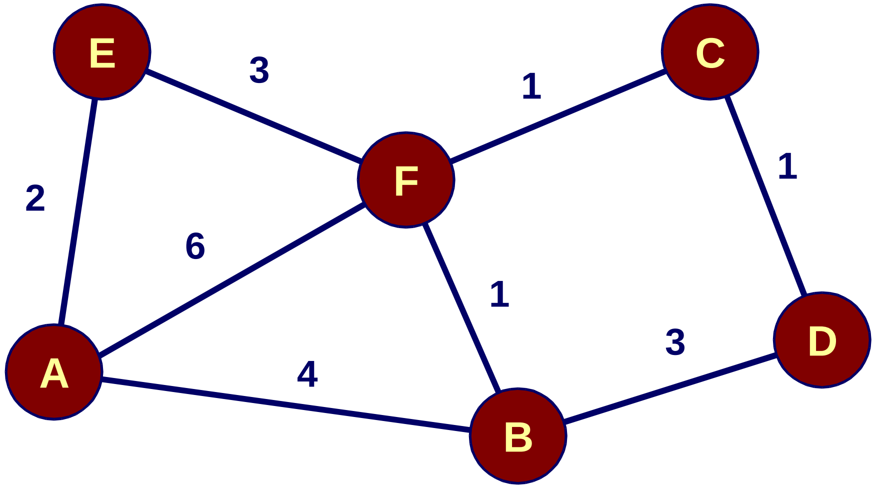


Table for A			Table for B			Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A	A	6	F	A	7	B	A	2	A	A	5	B
B	4	B	B	0	B	B	2	F	B	3	B	B	4	F	B	1	B
C	6	E	C	2	F	C	0	C	C	1	C	C	4	F	C	1	C
D	7	B	D	3	D	D	1	D	D	0	D	D	5	F	D	2	C
E	2	E	E	4	F	E	4	F	E	5	C	E	0	E	E	3	E
F	5	E	F	1	F	F	1	F	F	2	C	F	3	F	F	0	F

Asynchronous Version

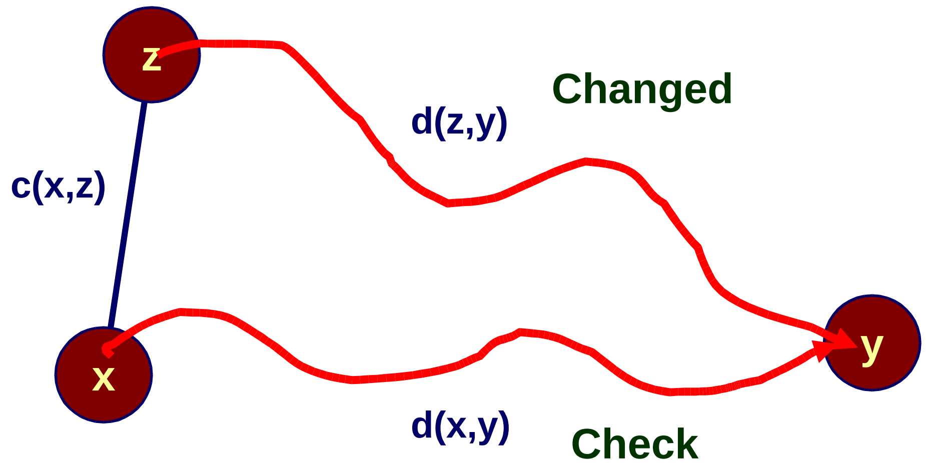
- Can be performed without any centralized control

Repeat

Choose arbitary x, y, z

$$d(x,y) \leftarrow \text{Update}(x,y,z)$$

Until Converge



How to "choose arbitrarily"

When value of d(z,y) changes, send message to all neighbors x

Convergence Properties

Ordering

- Let D denote values d(u,v) for all u & v
- Say D′ ≤ D when d′(u,v) ≤ d(u,v) for all u & v

Effect of Any Updating Step

- Describe as D′ ← Update(D,x,y,z)
- Gives new values D′ such that D′ ≤ D
  - "Monotonic"
- Values cannot go below 0

Implications

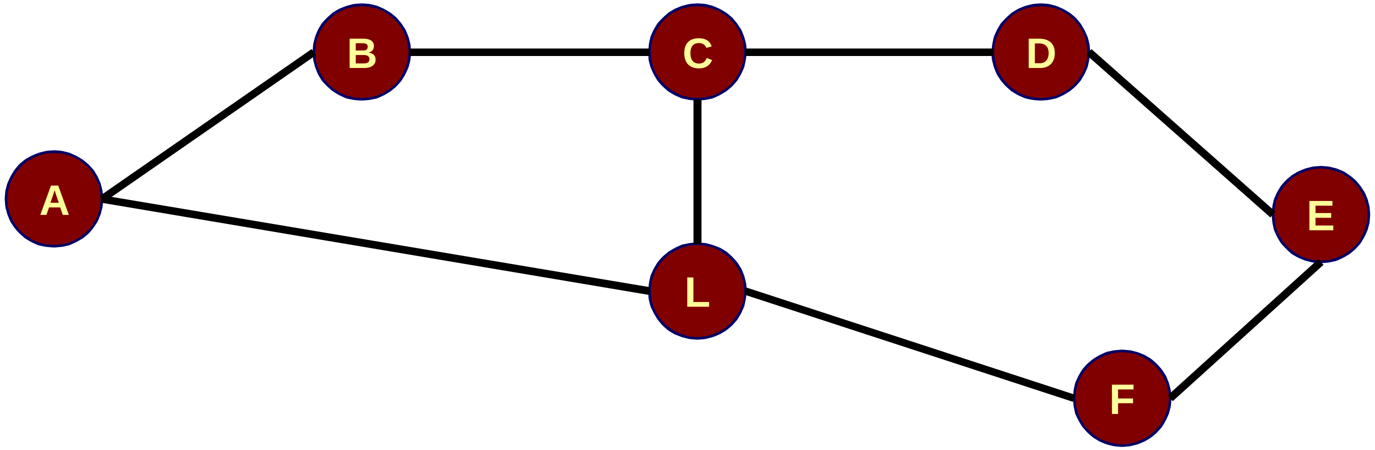
- Converges to unique "minimum fixed point" cost matrix D*
  - "Fixed point" means D* = Update(D*,x,y,z) for all x, y, & z
  - Tarski Fixed Point Theorem
  - (Multiple path-sets can have same minimal-cost D*)

Convergence Properties

Note

- Convergence proof is for ***static*** topology
  - No new nodes
  - No link-cost changes
  - No node failures

Asynchronous = Unpredictable



- **Consider A-F paths**
- **Two ways for A-F path to converge**
  - E tells D, D tells C, C tells B, B tells A; ***then*** L tells A, C
    - C learns C-D-E-F before learning C-L-F
    - A learns A-B-C-D-E-F before learning A-L-F
  - L tells A,C "right away"
    - A, C learn optimal route immediately

What if Node Fails?

- What if C crashes?
- F & D will stop receiving updates
- F & D will declare their links to C "down"
- B will learn "later"

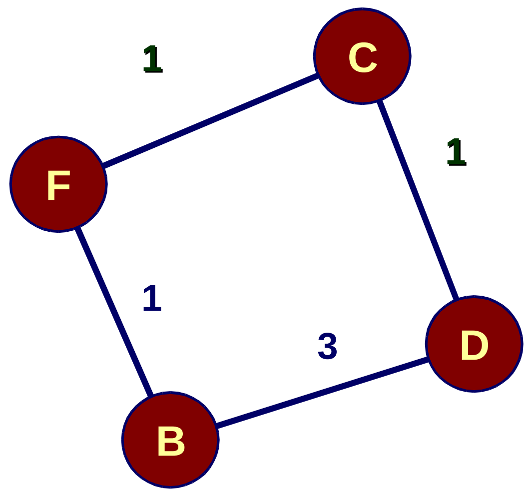


Table for B		
Dst	Cst	Hop
C	2	F

Table for D		
Dst	Cst	Hop
C	2	C

Table for F		
Dst	Cst	Hop
C	2	C

Link Failure

- **Set entries to ∞**
- **Iterate**

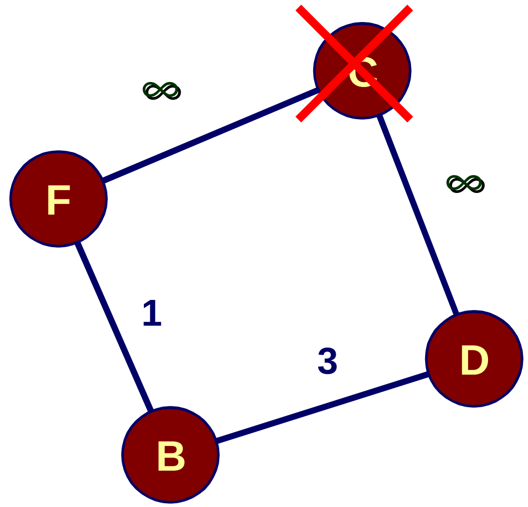


Table for D		
Dst	Cst	Hop
C	∞	–

Table for F		
Dst	Cst	Hop
C	∞	–

Failing Node Iterations

- Stale entry in B propagates to D & F

What Happened?

- Algorithm converges
- Can get wrong values

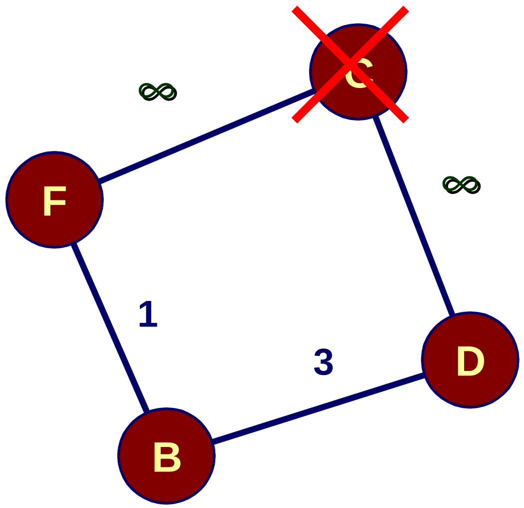


Table for B		
Dst	Cst	Hop
C	2	F

Table for D		
Dst	Cst	Hop
C	∞	–

Table for F		
Dst	Cst	Hop
C	∞	–

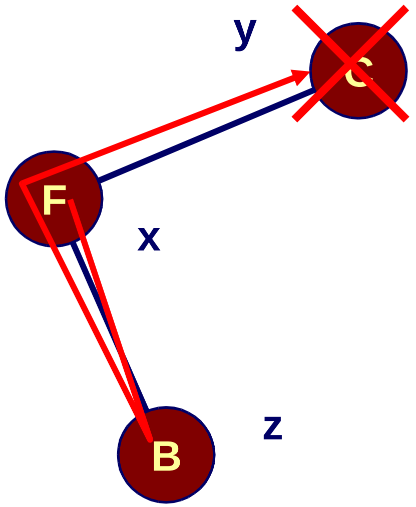
Better Route

Table for D		
Dst	Cst	Hop
C	5	B

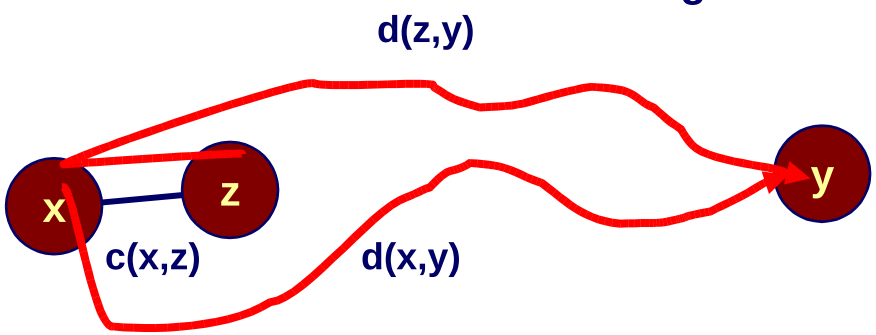
Better Route

Table for F		
Dst	Cst	Hop
C	3	B

Revised Update Rule #1



Creating circular path



Sometimes called "Split Horizon Rule"

Update(router=x, dest=y, peer=z)

- **d ← c(x,z) + d(z,y)** **# Cost of path from x to y with first hop z**
- **if d < d(x,y) & x ≠ nexthop(z,y)**

  # Found better path

  return d,z
- **else**

  return d(x,y), nexthop(x,y)

Iterations with Revision #2

- **Stale entry in B still propagates**

What Happened?

- **Algorithm converges**
  - **Tarski's theorem**
- **Can get wrong values**

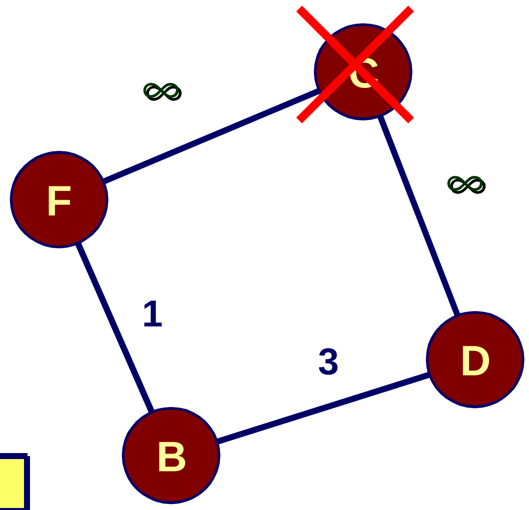


Table for B		
Dst	Cst	Hop
C	2	F

Table for D		
Dst	Cst	Hop
C	∞	–

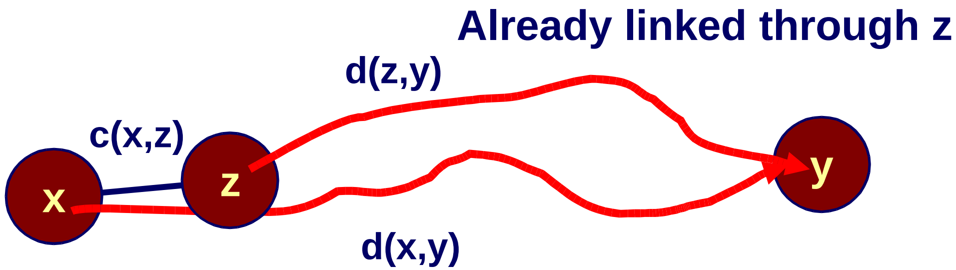
Table for F		
Dst	Cst	Hop
C	∞	–

Better Route

Table for D		
Dst	Cst	Hop
C	5	B

No Change

Revised Update Rule #2



Update(router=x, dest=y, peer=z)

- $d \leftarrow c(x,z) + d(z,y)$ # Cost of path from x to y with first hop z
- if nexthop(x,y) = z || # Forced update, regardless of cost

  $(d < d(x,y)$ & $x \neq$ nexthop$(z,y))$

  # Forced update or found better path

  return d,z
- else

  return d(x,y), nexthop(x,y)

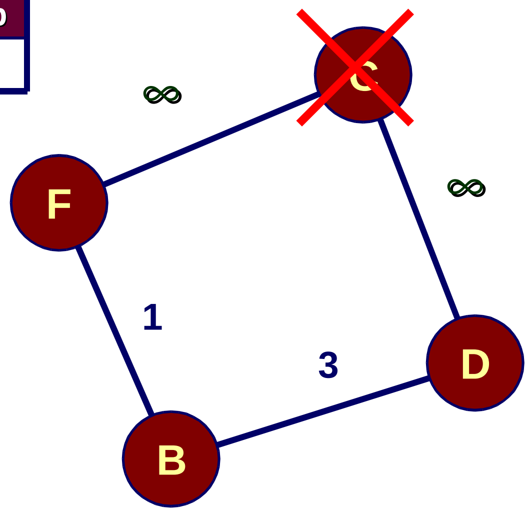
Iterations with Revision #2

Table for B		
Dst	Cst	Hop
C	2	F

Table for D		
Dst	Cst	Hop
C	∞	–

Table for F		
Dst	Cst	Hop
C	∞	–

No Change



Better Route

Table for D		
Dst	Cst	Hop
C	5	B

Forced Update

Table for B		
Dst	Cst	Hop
C	∞	–

Forced Update

Table for D		
Dst	Cst	Hop
C	∞	–

- Forced updates will eliminate false entries

Scary Feature

- Forced update rule violates monotonicity
  - Increases d(x,y)

Convergence Problems

Table for A			Table for B			Table for D			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
C	7	F	C	8	A	C	9	B	C	1	C

Forced Update

Table for F		
Dst	Cst	Hop
C	∞	–

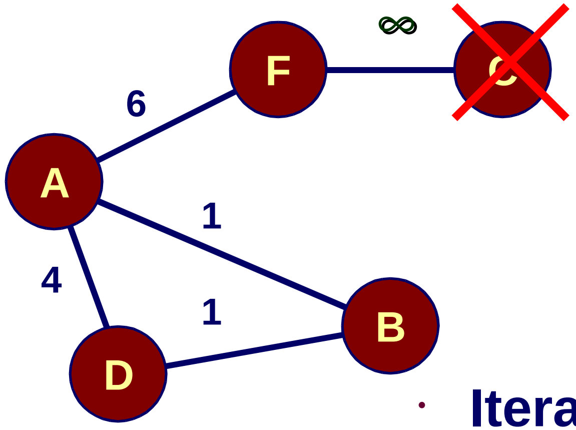


Table for A		
Dst	Cst	Hop
C	∞	–

Forced Update

Table for A		
Dst	Cst	Hop
C	13	D

Better Route

Forced Update

Table for B		
Dst	Cst	Hop
C	14	A

Forced Update

Table for D		
Dst	Cst	Hop
C	15	B

Table for A		
Dst	Cst	Hop
C	19	D

Forced Update

•
•
•

- Iterations don’t converge
- “Count to infinity”

Solution

- Make “infinity” smaller
- What is upper bound on maximum path length?

Routing Information Protocol (RIP)

- Earliest IP routing protocol (1982 BSD)
  - Ideas in first Arpanet protocols (late 60's)
- Current standard is version 2 (RFC 2453)

Features

- Every link has cost 1
- "Infinity" = 16
  - Limits to networks where everything reachable within 15 hops
  - Appropriate for "campus" networks

Sending Updates

- Every router listens for updates on UDP port 520
- RIP message can contain entries for up to 25 table entries

RIP Updates

Initial

- When router first starts, asks for copy of table for every neighbor
- Uses it to iteratively generate own table

Periodic

- Every 30 seconds, router sends copy of its table to each neighbor
- Neighbors use to iteratively update their tables

Triggered

- When every entry changes, send copy of entry to neighbors
  - Except for one causing update (split horizon rule)
- Neighbors use to update their tables

RIP Staleness / Oscillation Control

"Count to infinity"

- ...quick "for small values of infinity"

Route Timer

- Every route has timeout limit of 180 seconds
  - Reached when haven't received update from next hop for 6 periods
- If not updated, set to infinity

Behavior

- When router or link fails, can take minutes to stabilize
- Lots of subtlety to get good implementation (see RFCs).

Features of Distributed Algorithms

Desirable in Network Setting

- Every node operates in purely local way
  - No central control or global synchronization
- Only communication between direct neighbors

Not Difficult to Handle Static System

- Monotonicity guarantees convergence

Difficult in Dynamically-Changing System

- Anything that reduces link cost OK
  - Iterations will converge to reflect reduced costs
- Anything that increases link cost problematic
  - Iterations will converge, but possibly to wrong values
  - Changing update rule can lead to convergence problems
    - » Violate monotonicity