

CSC4200/5200 – COMPUTER NETWORKING

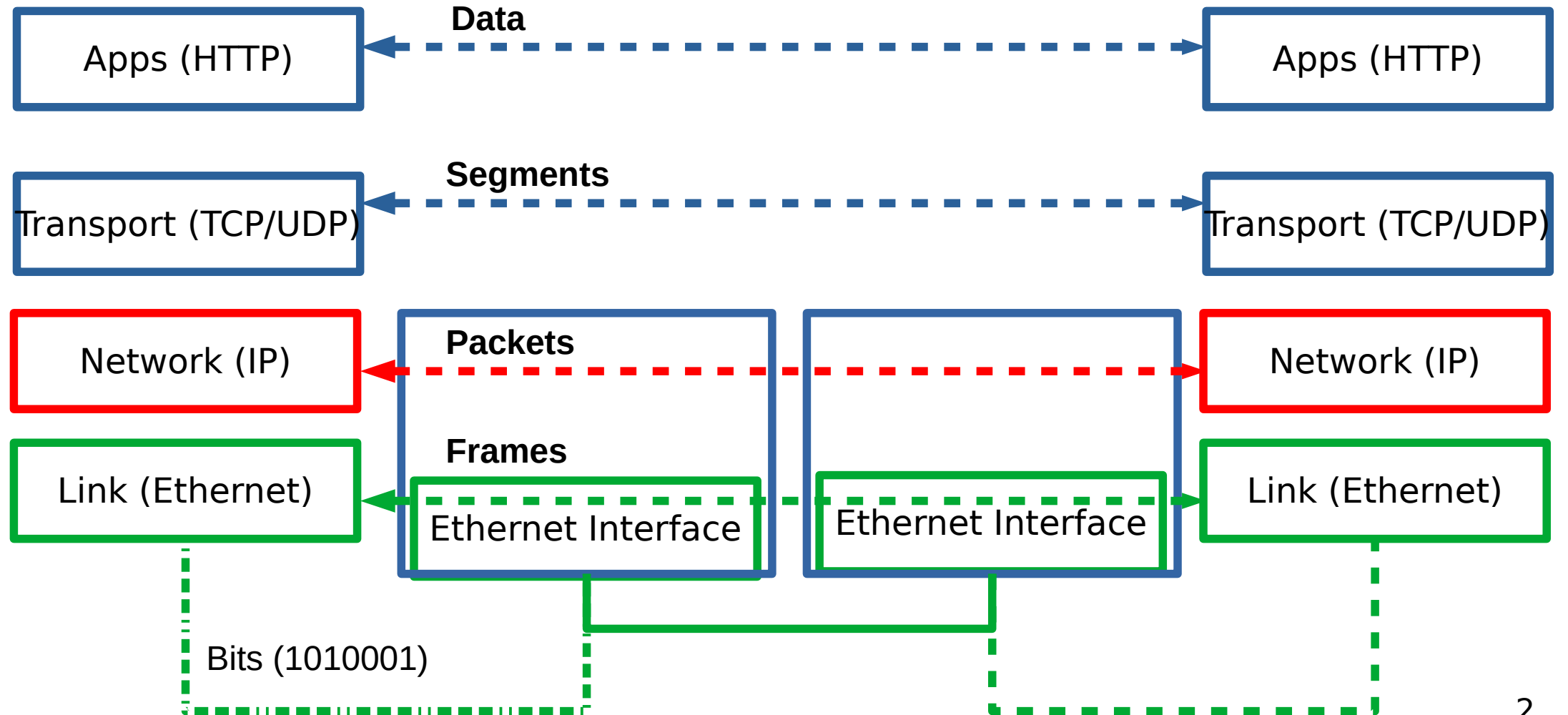
Instructor: Susmit Shannigrahi

ROUTING - CONTINUED

sshannigrahi@tnitech.edu

GTA: dereddick42@students.tnitech.edu





So far...

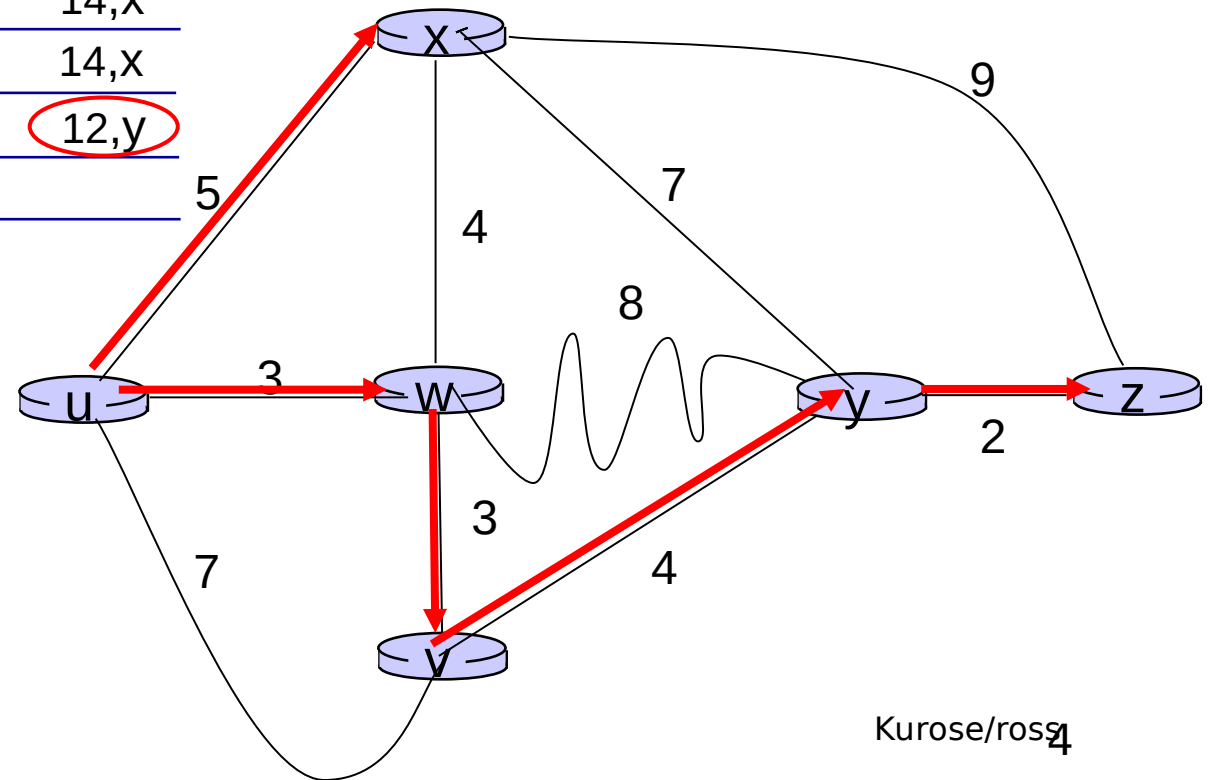
- Routing – Link state

Dijkstra's algorithm: example

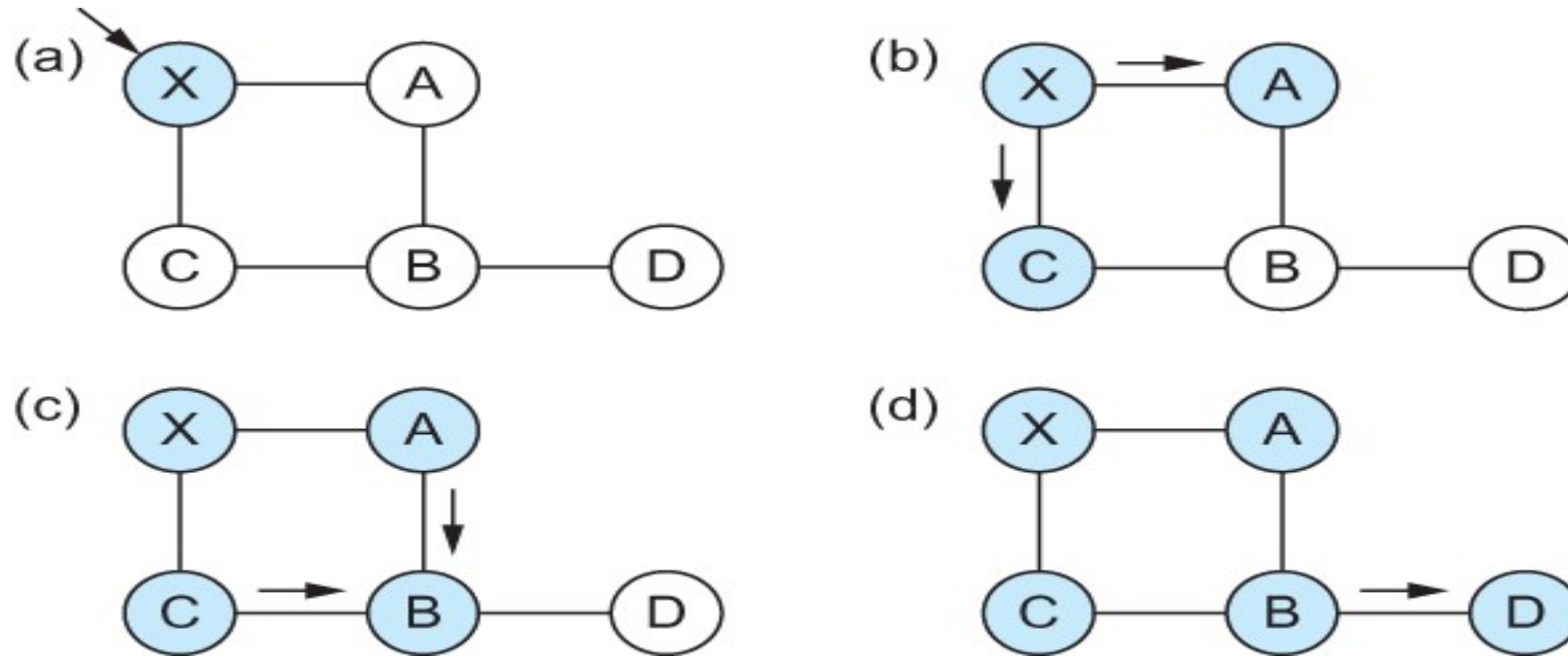
Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist (can be broken arbitrarily)



Link State Routing – controlled flooding



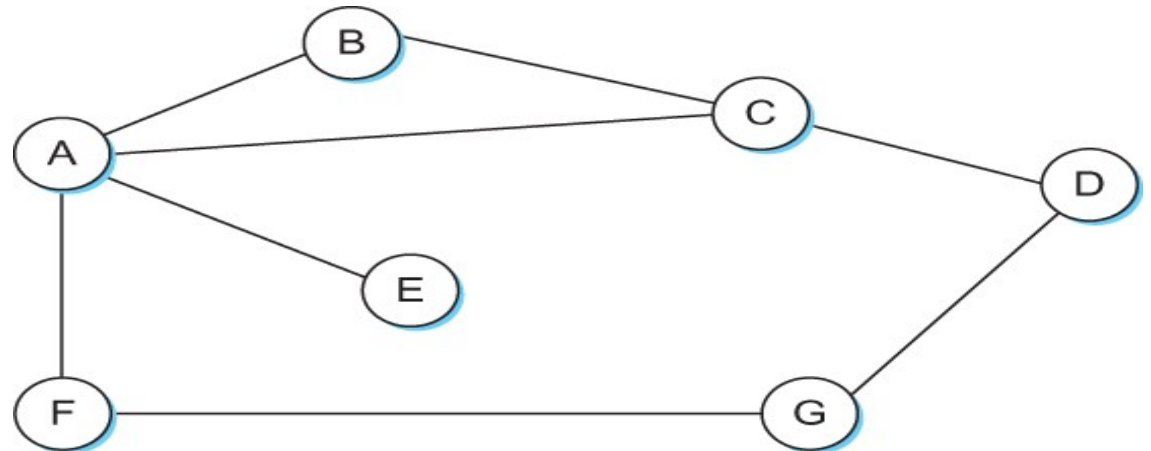
Flooding of link-state packets. (a) LSP arrives at node X; (b) X floods LSP to A and C; (c) A and C flood LSP to B (but not X); (d) flooding is complete

Example

- OSPF – Open shortest path first
- Problems -
 - does not scale!
 - Overhead
 - Reliable flooding may not be reliable

Distance Vector

- Each node has an one dimensional array (a vector) containing the “distances” (costs) to all other nodes
- Each node knows the cost to neighbors
- Each node distributes that vector to its immediate neighbors



Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y) :=$ cost of least-cost path from x to y

then

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

cost to neighbor v

cost from neighbor v to destination y

min taken over all neighbors v of x

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x
table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

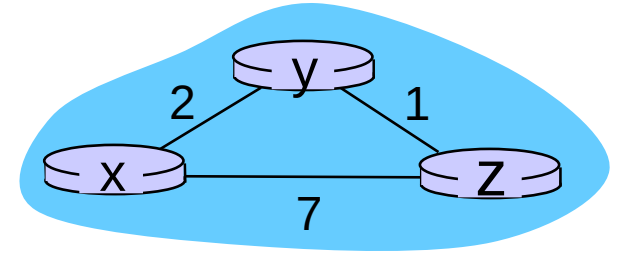
node y
table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

node z
table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0



time

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x
table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

node y
table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

node z
table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	7	1	0

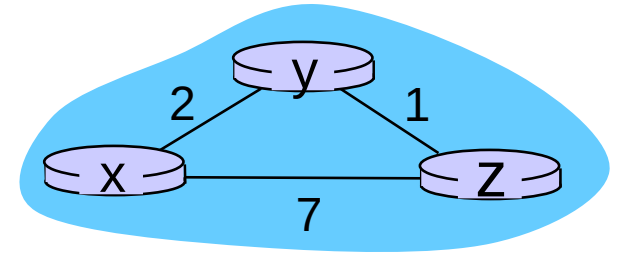
		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0

time



Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- ❖ under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm

iterative, asynchronous:

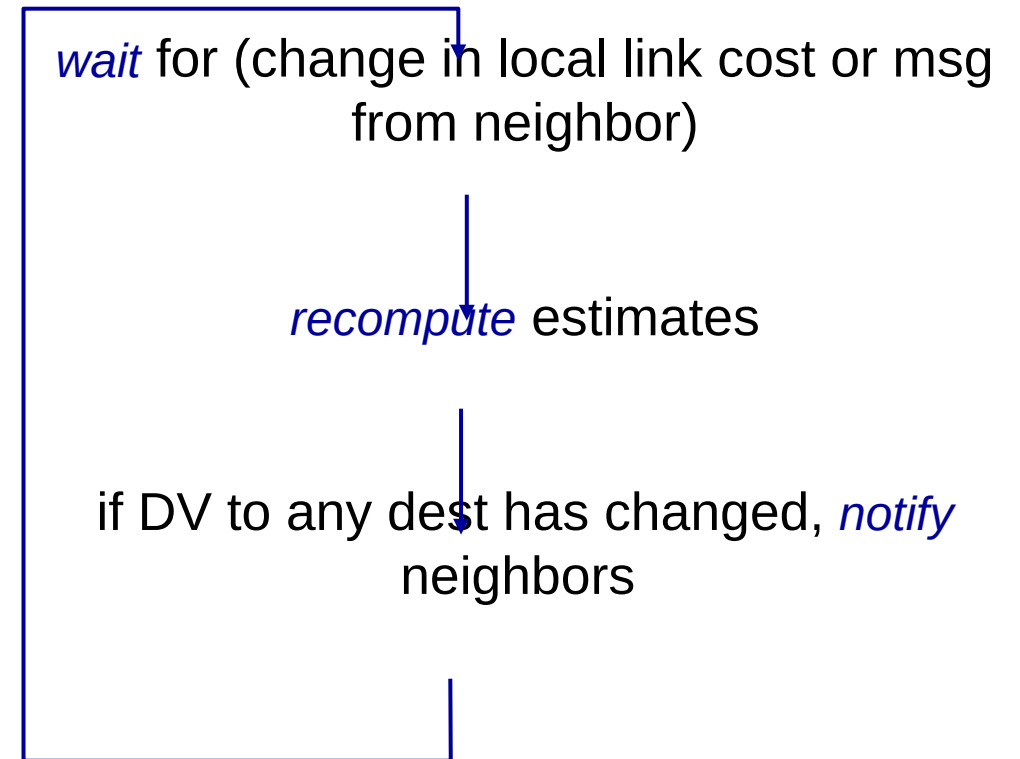
each local iteration caused by:

- local link cost change
- DV update message from neighbor

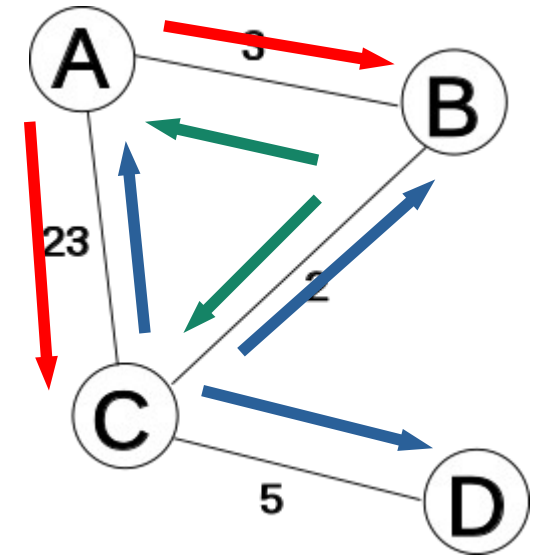
distributed:

- each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary

each node:



Distance vector algorithm



T=4

from A	via A	via B	via C	via D
to A				
to B		3	25	
to C		5	23	
to D		10	28	

from B	via A	via B	via C	via D
to A	3		7	
to B				
to C	8		2	
to D	13		7	

from C	via A	via B	via C	via D
to A	23	5		15
to B	26	2		12
to C				
to D	33	9		5

from D	via A	via B	via C	via D
to A			10	
to B			7	
to C			5	
to D				

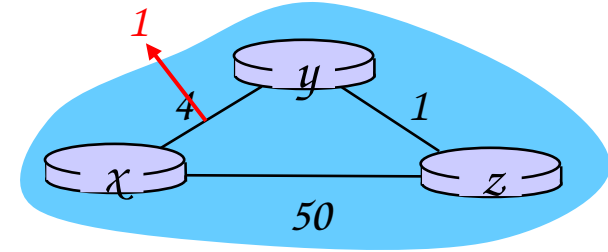
Wikipedia

Initial distances stored at each node (global view)

Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors



“good
news
travels
fast”

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

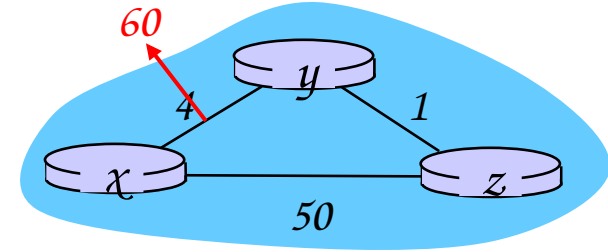
t_1 : z receives update from y , updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z 's update, updates its distance table. y 's least costs do *not* change, so y does *not* send a message to z .

Distance vector: link cost changes

link cost changes:

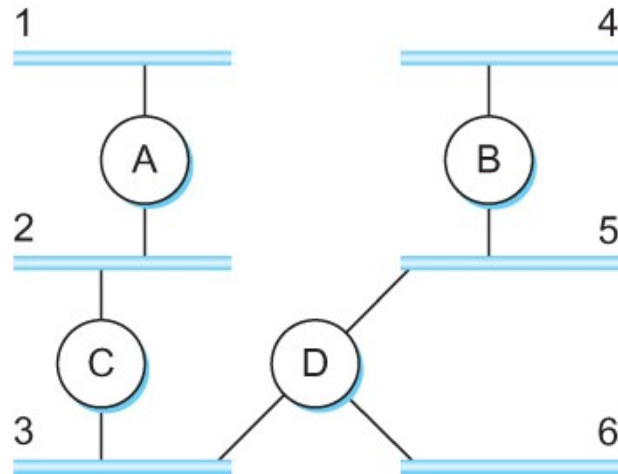
- ❖ node detects local link cost change
- ❖ *bad news travels slow* - “count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes



poisoned reverse:

- ❖ If Z routes through Y to get to X :
 - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- ❖ will this completely solve count to infinity problem?

Distance vector: Routing Information Protocol (RIP)



Max hops = 16

0	8	16	31
Command	Version	Must be zero	
Family of net 1		Route Tags	
Address prefix of net 1			
Mask of net 1			
Distance to net 1			
Family of net 2		Route Tags	
Address prefix of net 2			
Mask of net 2			
Distance to net 2			

Comparison of LS and DV algorithms

message complexity

- **LS:** with n nodes, E links, $O(nE)$ msgs sent
- **DV:** exchange between neighbors only
 - convergence time varies

speed of convergence

- **LS:** $O(n^2)$ algorithm requires $O(nE)$ msgs
 - may have oscillations
- **DV:** convergence time varies
 - may be routing loops
 - count-to-infinity problem

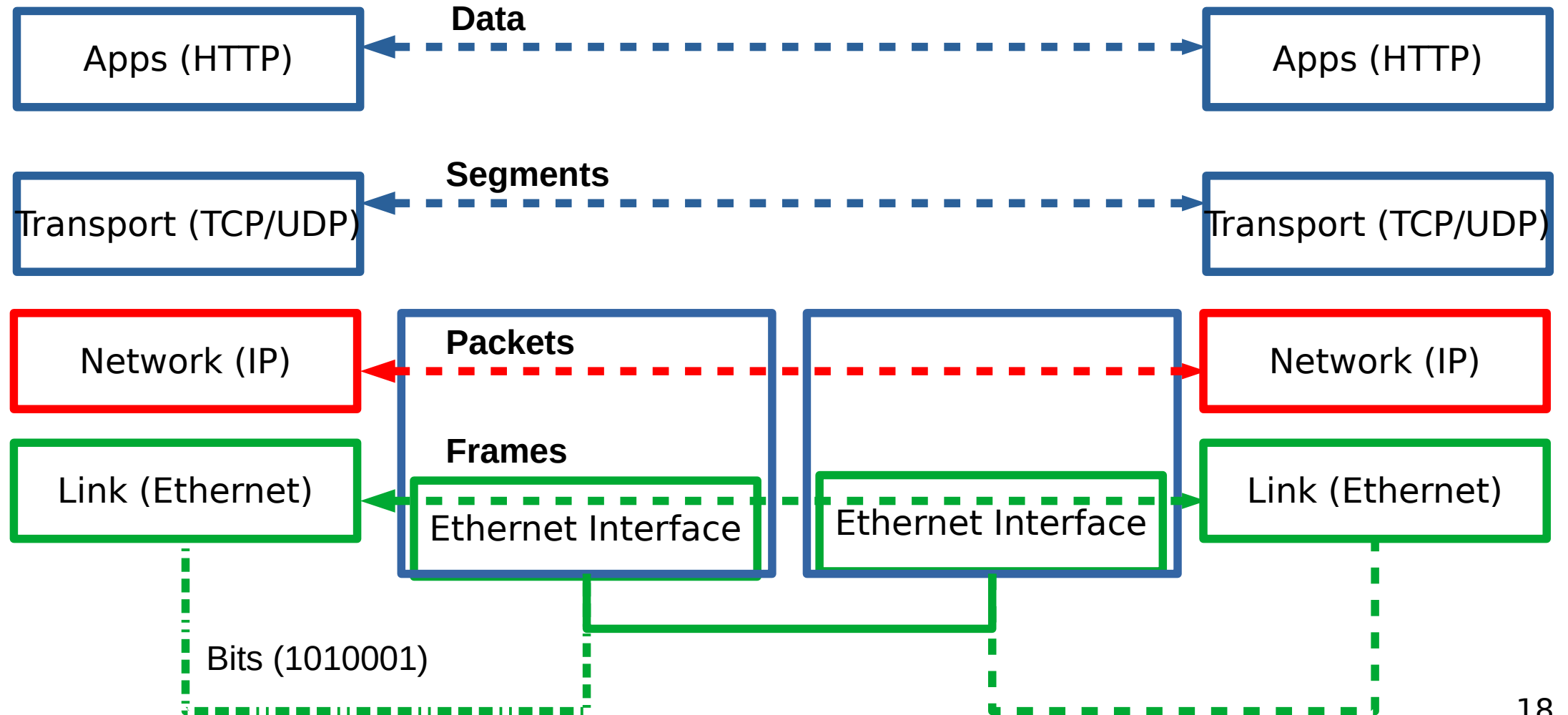
robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
 - error propagate thru network



Next Steps

How do we scale this routing to the Internet?