Last Lecture: Network Layer

1. *Design goals and issues* ✔
   - Debate around service model/design principle question
2. *Basic Routing Algorithms & Protocols*
   - Packet Forwarding
   - Shortest-Path Algorithms
   - Routing Protocols
3. *Addressing, Fragmentation and reassembly*
4. *Internet Routing Protocols and Inter-networking*
5. *Router design*
6. *Congestion Control, Quality of Service*
7. *More on the Internet's Network Layer*

This Lecture: Network Layer

1. *Design goals and issues*
   - Debate around service model/design principle question
2. *Routing and Forwarding* ✔
   - Packet Forwarding
   - Shortest-Path Algorithms
   - Routing Protocols
3. *Addressing, Fragmentation and reassembly*
4. *Internet Routing Protocols and Inter-networking*
5. *Router design*
6. *Congestion Control, Quality of Service*
7. *More on the Internet's Network Layer*

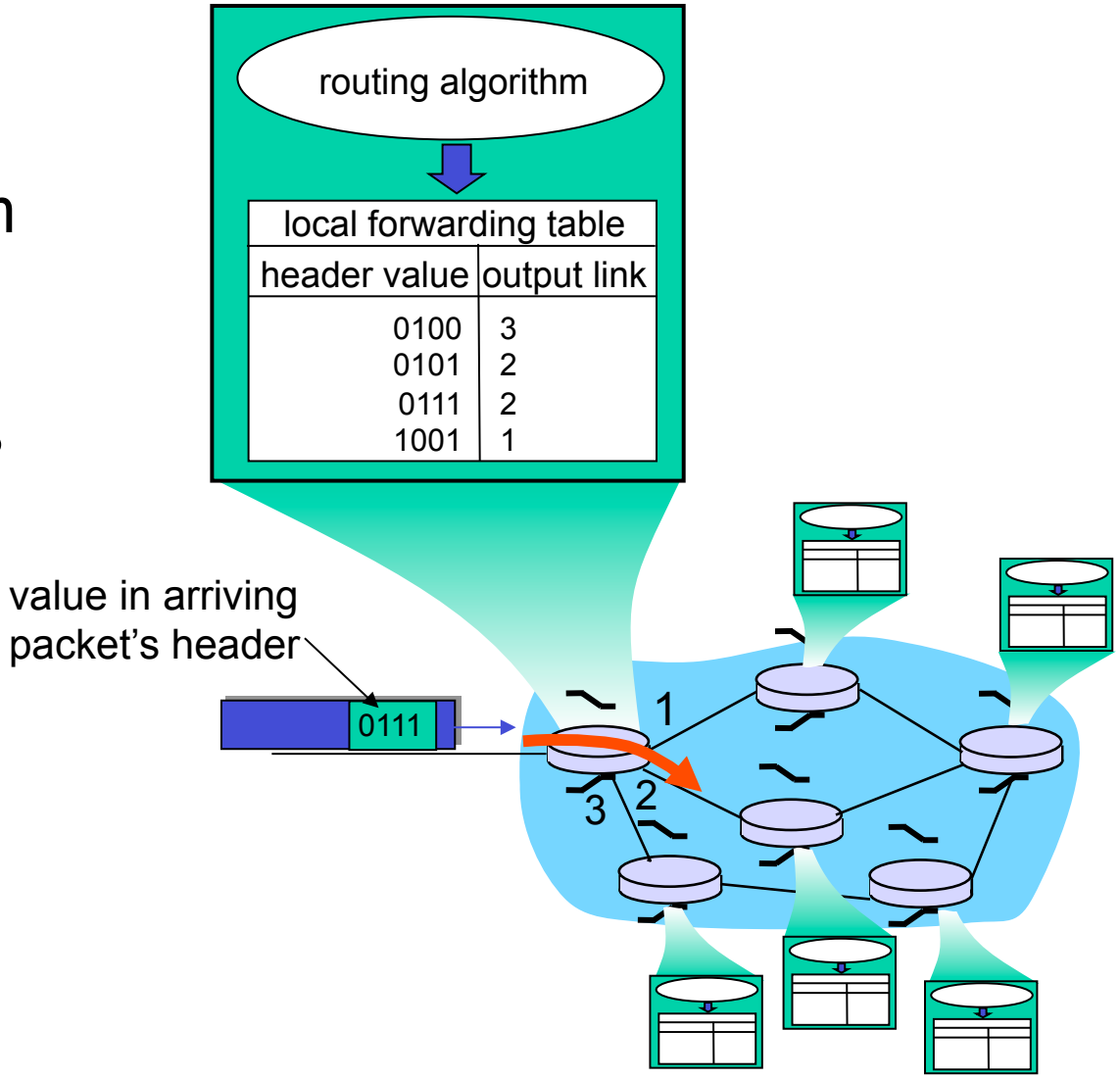
Routing and Forwarding

- Let's consider connection-oriented and connectionless services first
  - More on QoS and congestion control issues later

- Three main problems (on a packet switched net.)
  - *Forwarding*: how routers forward a packet
  - *Routing*: routers "learn" network topology (more later)
  - *Addressing*: identify hosts and routers (more later)

Routing and Forwarding

Routing algorithm determines values in the forwarding tables



1. Packet Forwarding

a) *Virtual Circuit* (connection-oriented)
   - Connection routed through network to set up state
   - Packets forwarded using connection state

b) *Datagram* (connectionless)
   - Routers keep next hop for destination
   - Packets carry destination address

c) *Source routing*:
   - Packet contains entire path to destination
   - Could done on either virtual circuit *or* datagram networks

a) Virtual Circuits

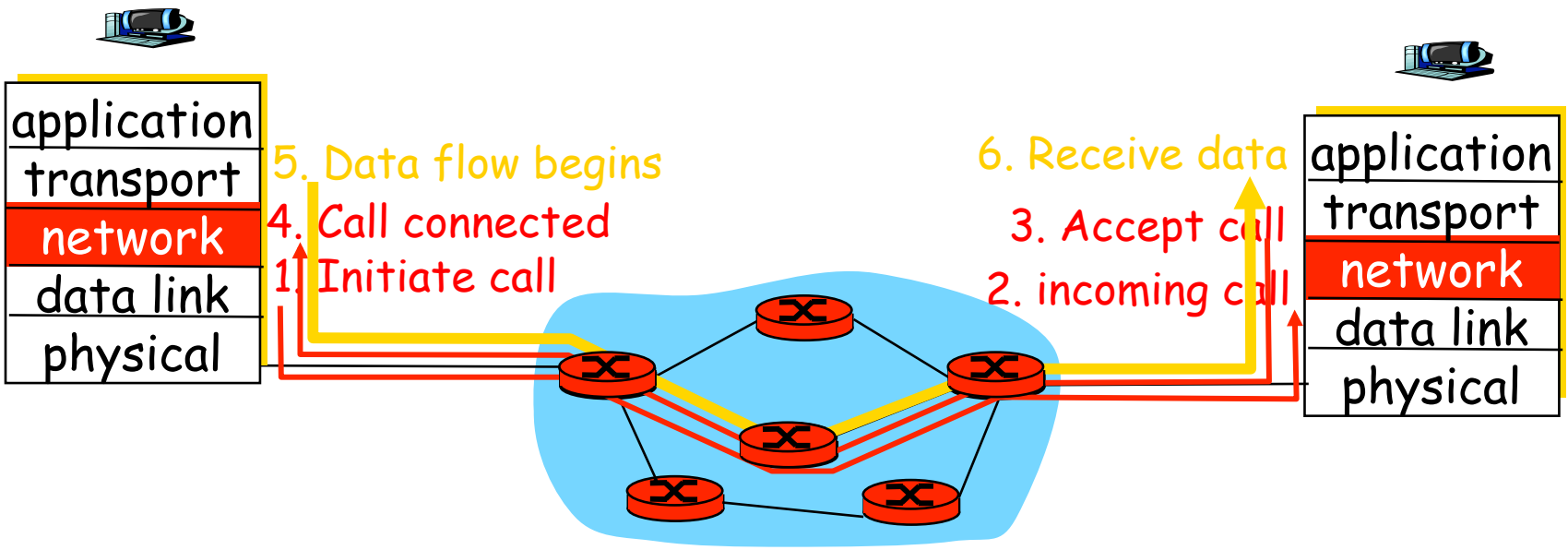
"Source-to-dest path behaves much like telephone circuit"

- Performance-wise
- Network actions along source-to-dest path

- Call setup *before* data can flow
- Call teardown *after* data flow
- Each packet carries VC ID (**not** destination host ID)
- *Every* router on source-destination path maintains "state" for each passing connection
- Link, router resources (bandwidth, buffers) may be *allocated* to VC
  - Get a circuit-like performance

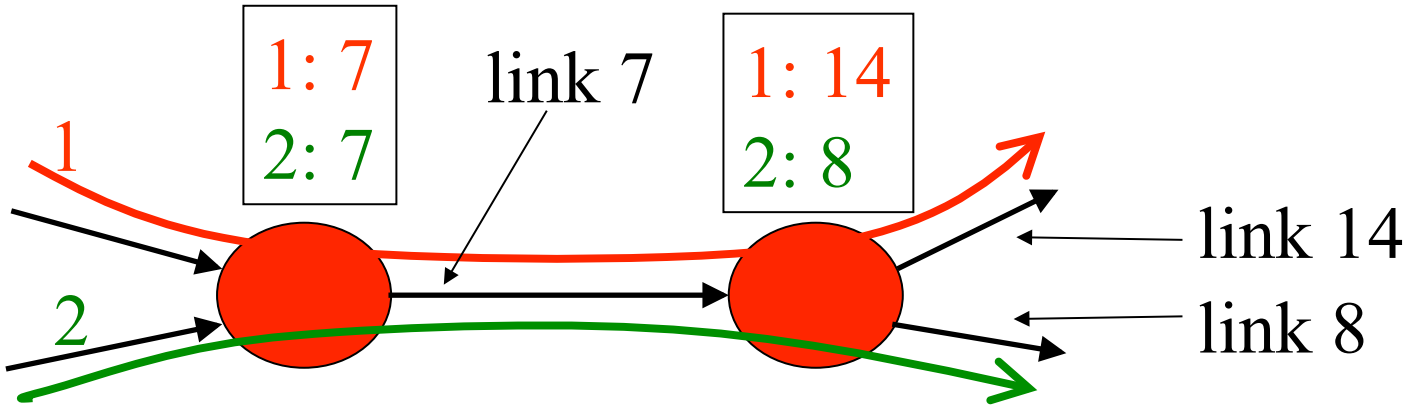
Virtual Circuits: Signaling Protocols

- Used to setup, maintain, teardown VC
- Used in ATM, frame-relay, X.25



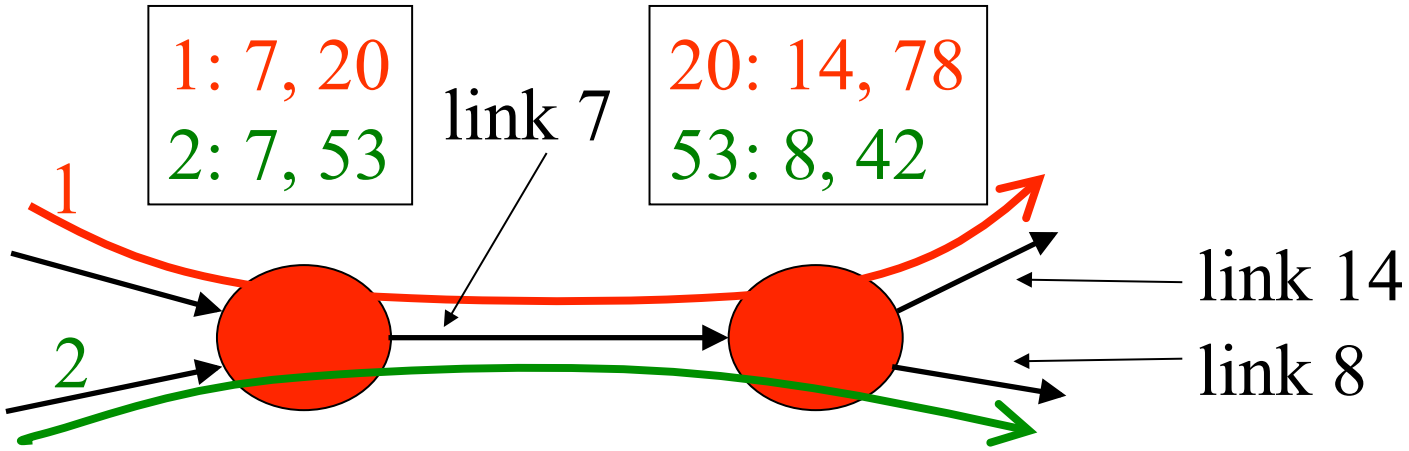
Virtual Circuit Identifier (VCI)

- Virtual Circuit Identifier (VC ID)
  - Source set-up: establish path for the VC
  - Switch: mapping VC ID to an outgoing link
  - Packet: fixed length label in the header

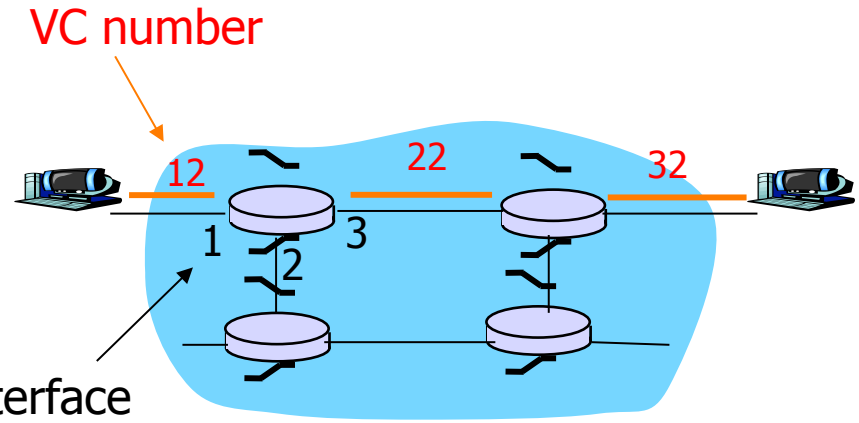


VCI Assignment

- Problem with single VCI for entire path
  - Use up VCI space very quickly
- Solution: label swapping
  - Map the VCI to a new value at each hop
  - Table has old ID, next link and new ID



Virtual Circuits: Forwarding Table



Forwarding table in northwest router:

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

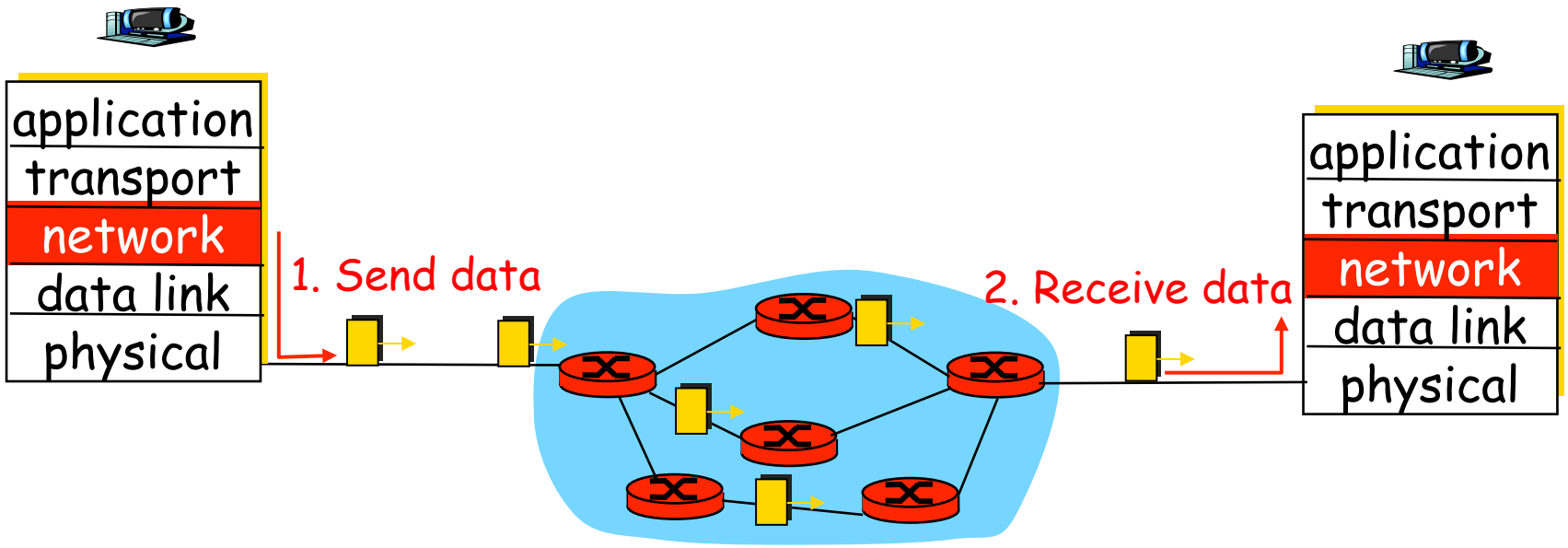
Routers maintain connection state information!

Virtual Circuits: Pros and Cons

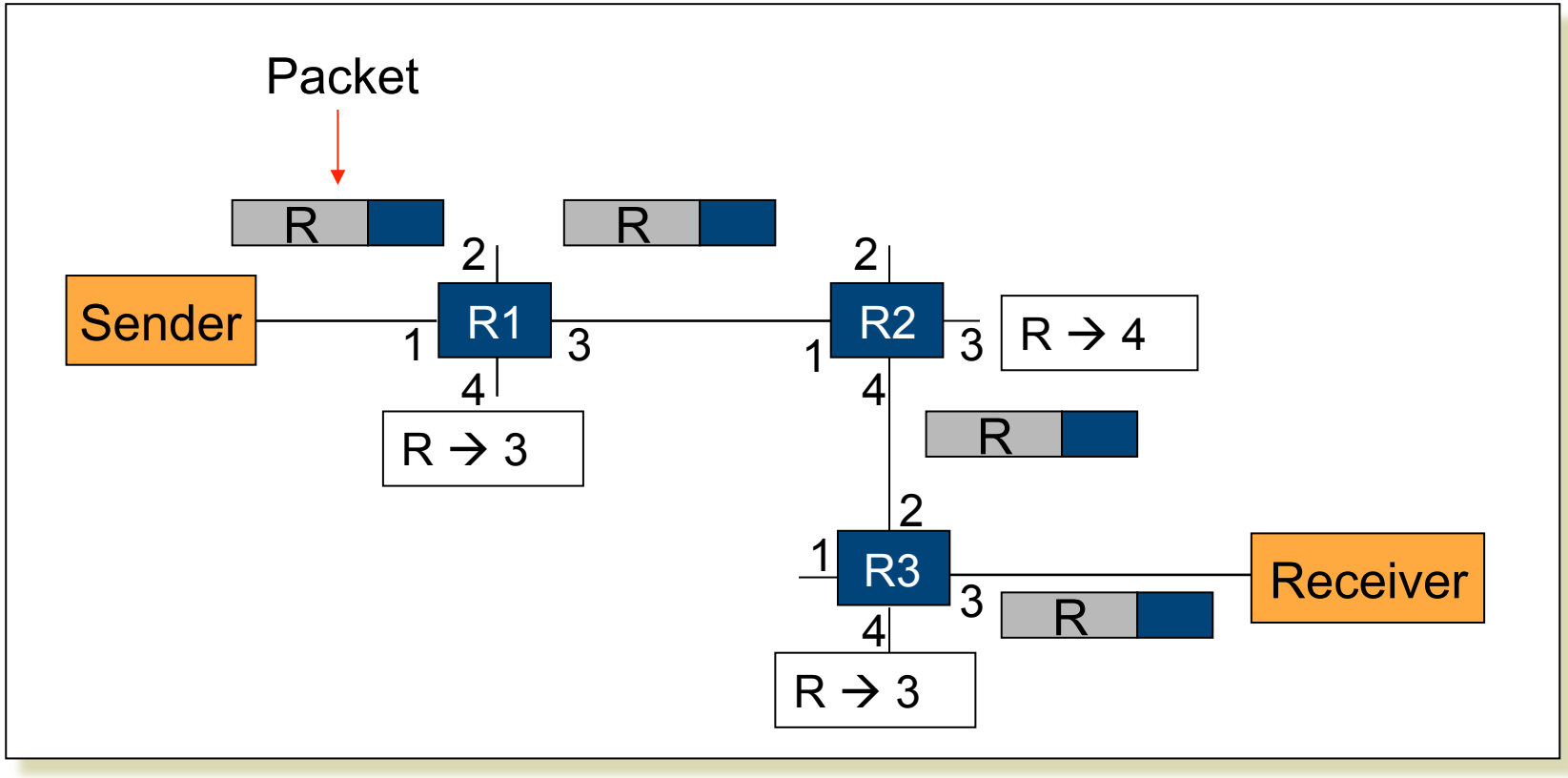
- Advantages
  - Efficient lookup (simple table lookup)
  - Can reserve bandwidth at connection setup
  - Easier for hardware implementations
- Disadvantages
  - Still need to route connection setup request
  - More complex failure recovery – must recreate connection state
- Typically used in
  - ATM – combined with fix sized cells
  - MPLS – tag switching for IP networks
  - X.25, FrameRelay, …

b) Datagram (the Internet Model)

- No call setup at network layer
- Routers: no "state" about end-to-end connections
  - No network-level concept of "connection"
- Packets forwarded using destination host address
  - Packets between same source-destination pair may take different paths



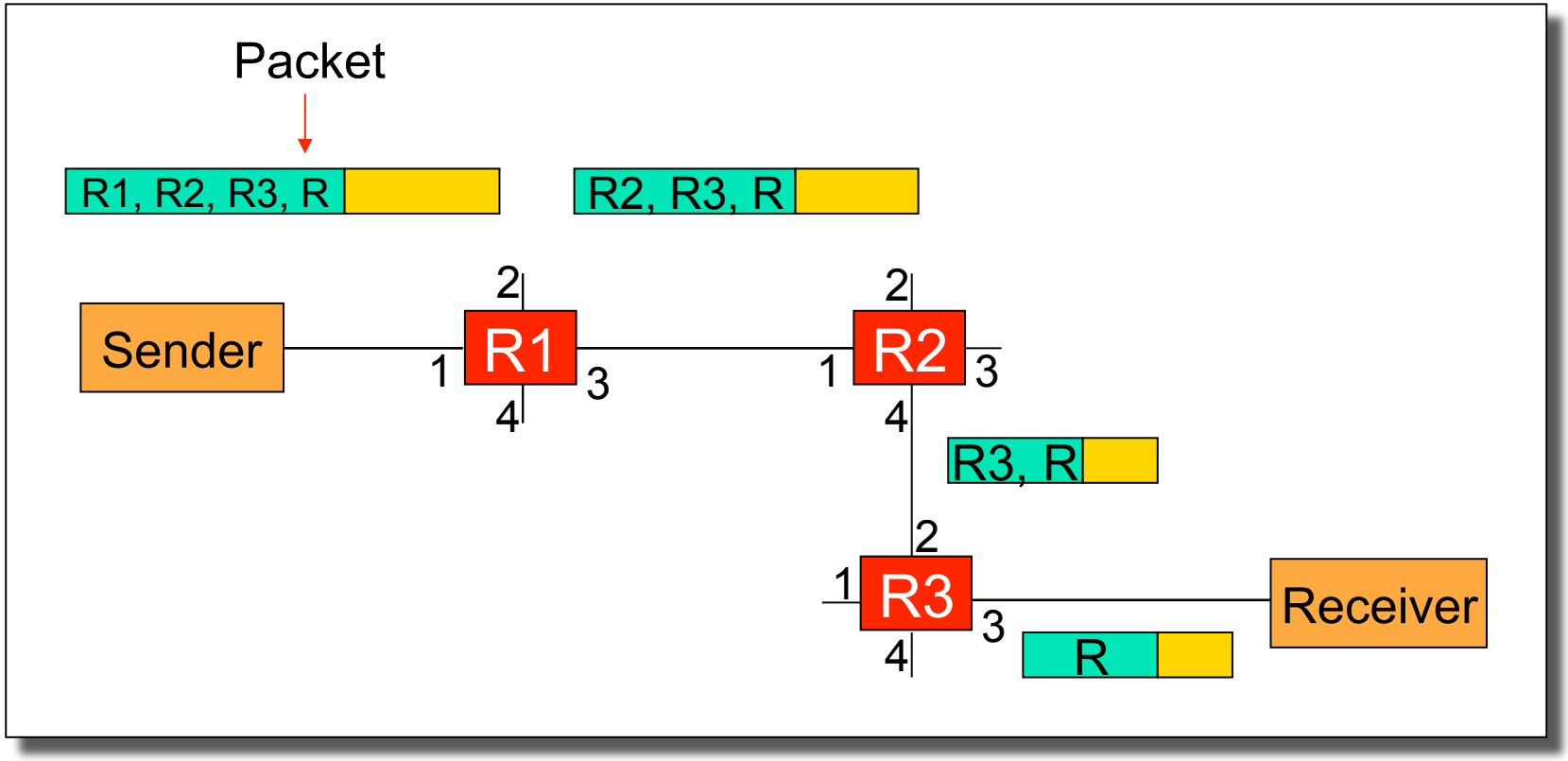
Packet Forwarding in Datagram Networks



Datagram: Pros and Cons

- Advantages
  - Stateless – simple error recovery
- Disadvantages
  - Every switch/router knows about every destination
    - Potentially *huge* tables
  - All packets to destination take same route
  - Need routing protocol to fill table
- Typical use: IP networks

c) Source Routing



Source Routing: Pros and Cons

- Advantages
  - Switches can be very simple and fast
- Disadvantages
  - Variable (unbounded) header size
  - Sources must know or discover topology (e.g., failures)
- Typical uses
  - Ad-hoc networks (DSR)
  - Machine room networks (Myrinet)

Packet Forwarding: Comparison

	Source Routing	Global Addresses	Virtual Circuits
Header Size	Worst	OK – Large address	Best
Router Table Size	None	Number of hosts (prefixes)	Number of circuits
Forward Overhead	Best	Prefix matching (Worst)	Pretty Good
Setup Overhead	None	None	Connection Setup
Error Recovery	Tell all hosts	Tell all routers	Tell all routers and Tear down circuit and re-route

Popular Network Layer Service Models

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

- Internet model being extended: Intserv, Diffserv

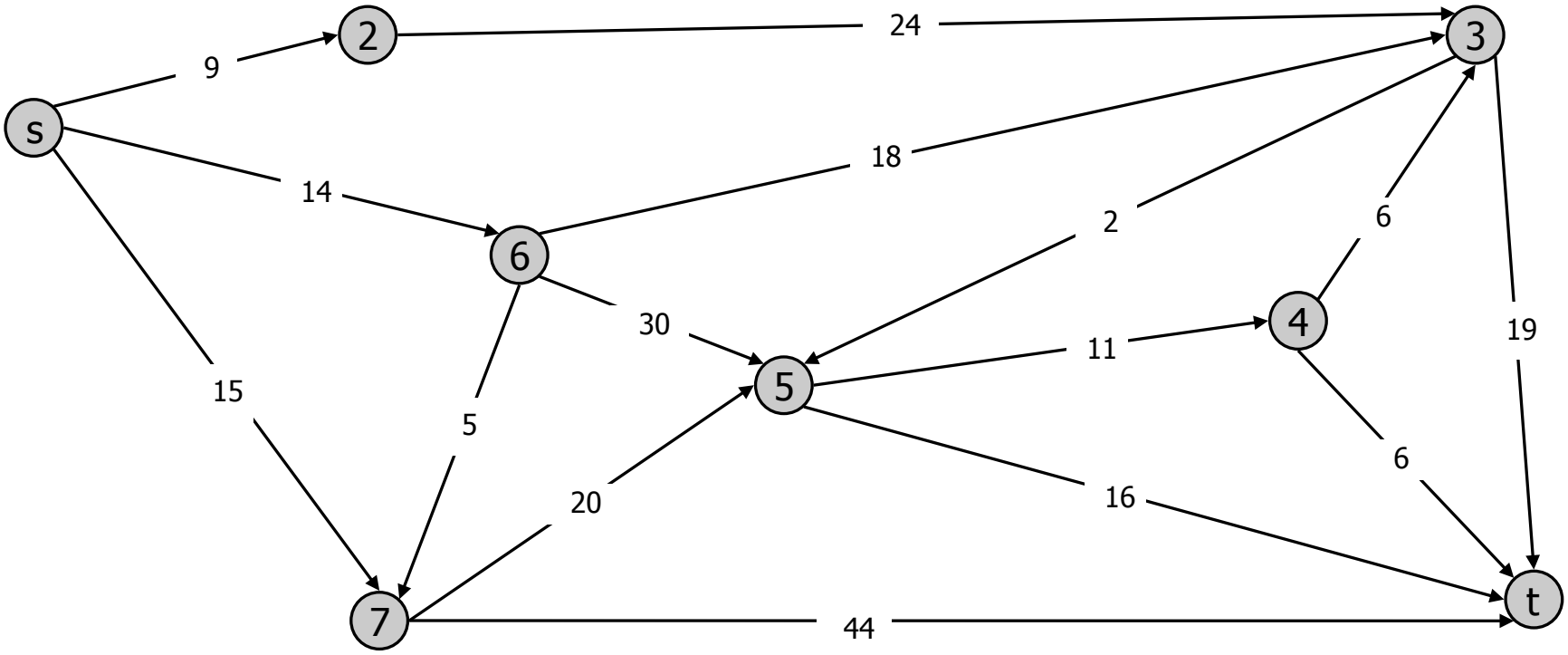
Datagram vs. VC Networks

Issue	**Datagram**	**VC**
Circuit Setup	NO	YES
State Info	Routers do not keep	Routers do keep
Routing	Packets routed independently	Packets' route chosen in advance
Effect of router failures	Very little	All VCs through router are finished
Quality of Service	Difficult	Easy if resources pre-allocated
Congestion Control	Difficult	Easy if resources pre-allocated
Utilization	High (statistical multiplexing)	Lower if resources pre-allocated
Network Neutrality	High	Low
Complexity	At edge	In the network

2. Shortest-Path Algorithms & Routing Protocols

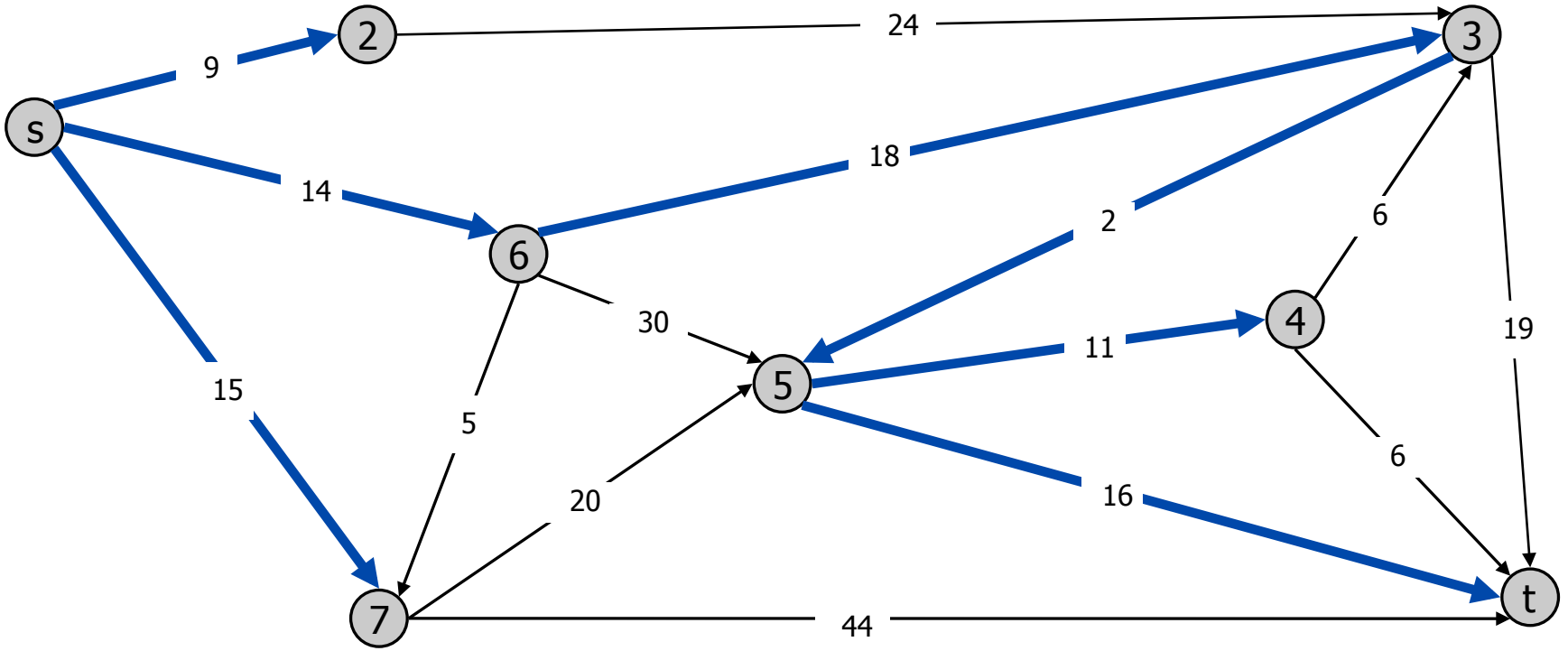
- Routing Algorithms
  - Find "shortest" path from a source to a destination
- Centralized: *shortest-path algorithms*
  - Dijkstra algorithm
  - Bellman-Ford algorithm
- Distributed: *routing protocols*
  - Link-state (distributive version of Dijkstra)
  - Distance-vector (distributive version of Bellman-Ford)
  - Path-vector (fixes some problems with DV)

Networks as Graphs



- Weights: combination of bandwidth, load, delay, …

Shortest Path Tree



- Directed tree, links go out from root to leaves
- The *unique* path from root to any node v is the shortest path from the root to v

Shortest Path Algorithms

- *Dijkstra*: a greedy algorithm
- *Bellman-Ford*: a dynamic programming algorithm

- *Input*:
  - Direct graph $G = (V,E)$
  - A weight function $w: E \rightarrow R^+$
  - A source s

- *Output*:
  - A shortest path tree rooted at s

Basic Data Structures and Functions

- To build the SPT, each node maintains two fields:
  - *p[v]*: the pointer to the parent of v in the tree
  - *c[v]*: the least cost from s to v

- **Init()**:
  - For each vertex v, $c[v] = \infty$, $p[v] =$ NIL
  - $c[s] = 0$

- **Improve(*u*,*v*)**, where (u,v) is a directed edge of G
  - if $c[v] > c[u] + w(u,v)$ then
    $c[v] = c[u] + w(u,v)$
    $p[v] = u$

Dijkstra Algorithm

- Init()
- T = empty-set
- while (T != V)
  - $u \leftarrow$ a vertex not in T with minimum $c[u]$
  - $T = T \cup \{u\}$ // add u to T
  - for each v not in T so that (u,v) is an edge
    Improve(u,v)
- Note:
  - If there are negative-weight edges, the algorithm is slightly different
  - It can also fail if there is a negative cycle

Tips and Tricks

Some Dijkstra's quotes:

- *"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."*

- "*Computer science is no more about computers than astronomy is about telescopes*."

- "*Object-oriented programming is an exceptionally bad idea which could only have originated in California.*"

Bellman-Ford Algorithm

- Init()
- For i=1 to $|V|$-1
  - for each edge (u,v) in G
    Improve(u,v)

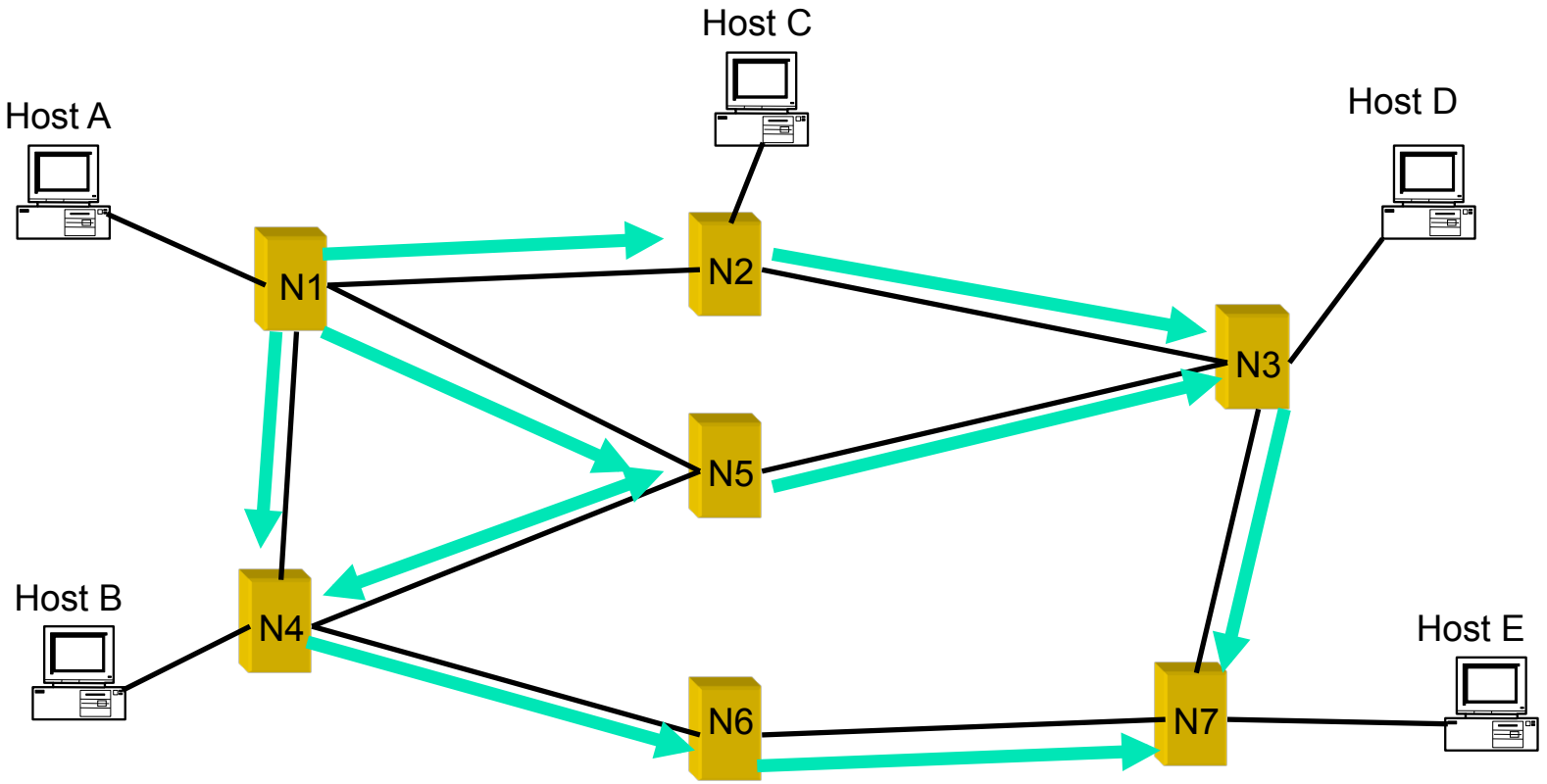
- Can be modified to allow negative weights and negative cycles

Link State and Distance Vector Basics

- *Link State (LS) Routing*
  - Each node periodically sends neighboring link costs to all nodes
  - Each node computes routing table separately
  - *Question*: what if two nodes compute two different SPTs (given the same graph?)
- *Distance Vector (DV) Routing*
  - Each node periodically sends distance estimates to all neighbors
  - Each node updates routing table accordingly
- Update periods often 30sec

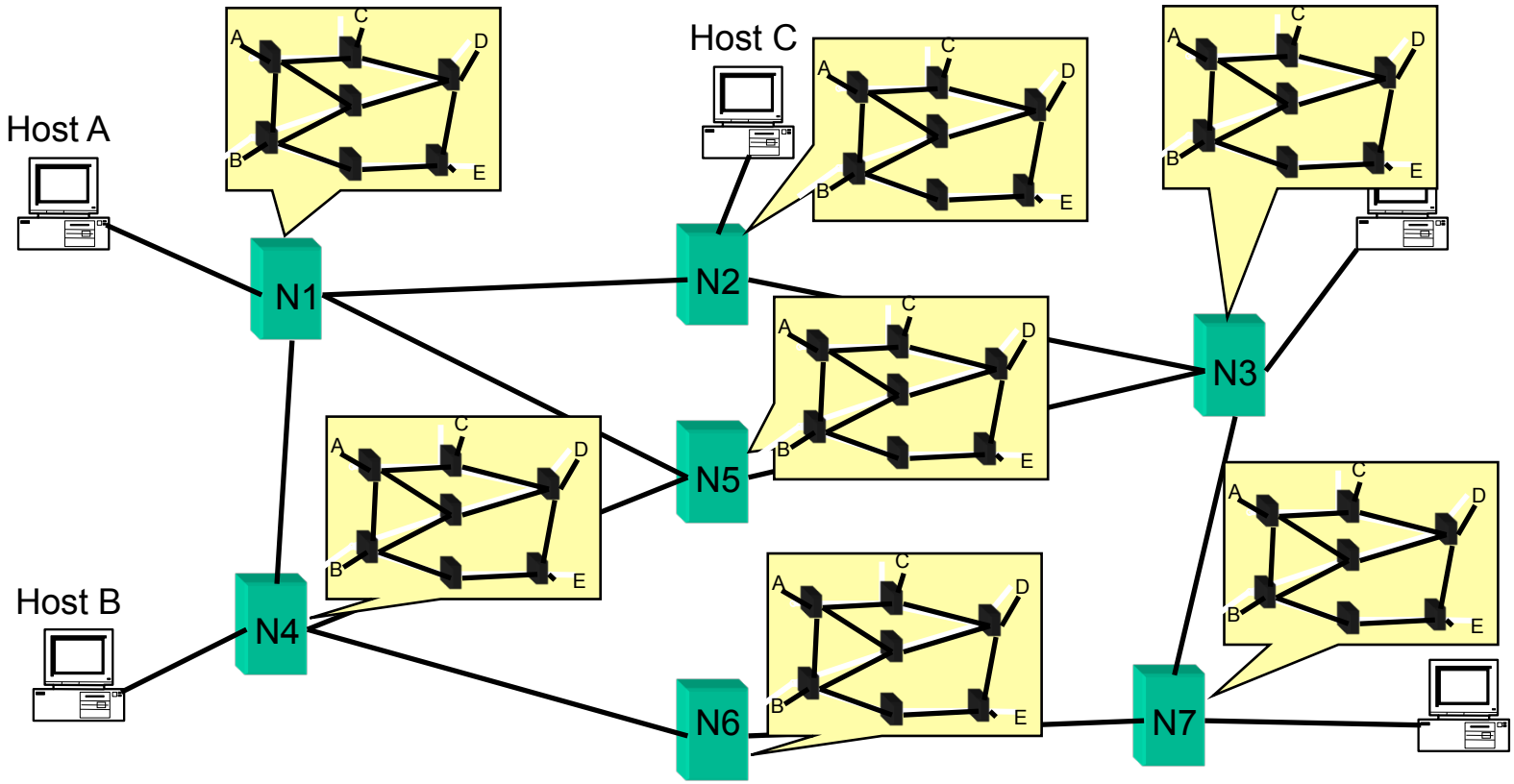
Link State: Control Traffic

- Each node floods its neighborhood information to all



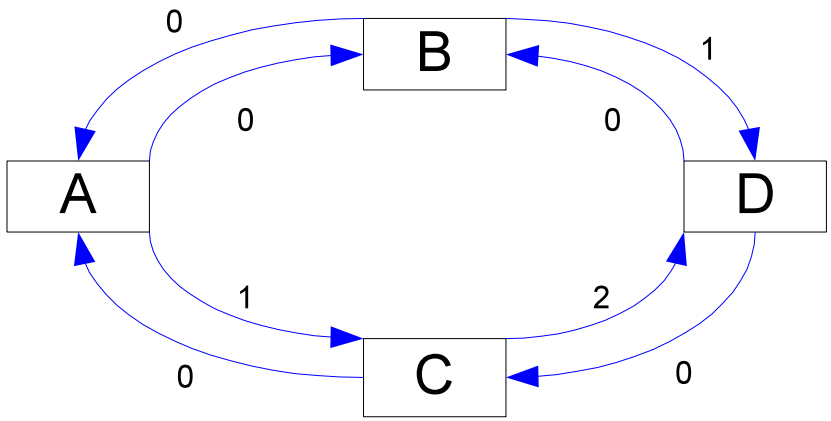
Link State: Node States

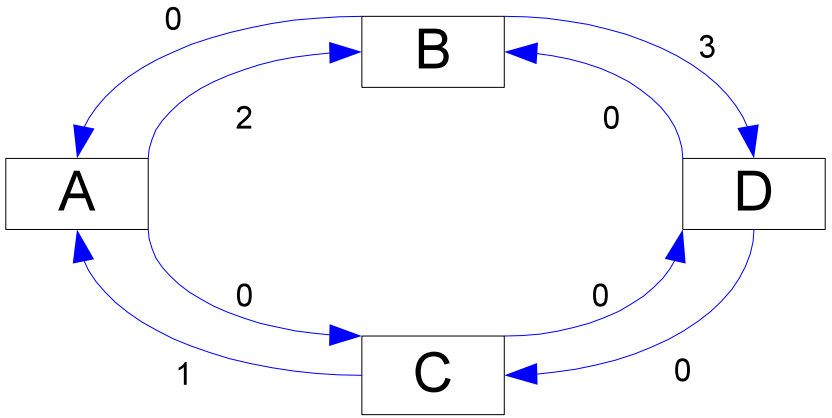
Net result: everyone knows the *entire* network topology → use Dijkstra to compute the shortest paths

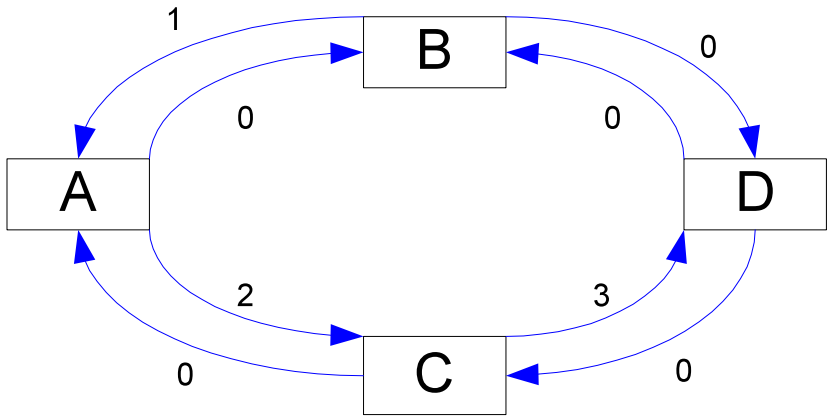


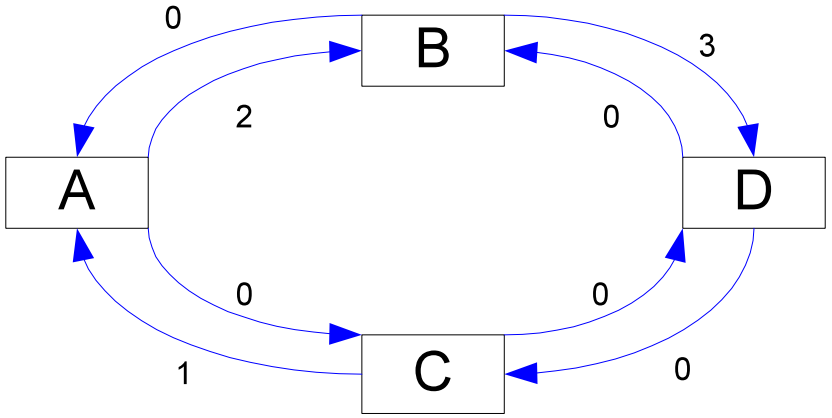
Oscillation Problem

- If link load is part of the cost









Link State's Problems

- *Topology information is flooded*
  - High bandwidth and storage overhead
  - Forces nodes to divulge sensitive information
- *Entire path computed locally per node*
  - High processing overhead in a large network
- *Minimizes some notion of total distance*
  - Works only if policy is shared and uniform
- *Typically used only inside an AS*
  - E.g., OSPF and IS-IS

Basic Distance Vector Protocol

- $c(x,v)$ = cost for direct link from x to v
  - Node x maintains costs of direct links $c(x,v)$
- $D_x(y)$ = estimate of least cost from x to y
  - Node x maintains *distance vector* $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x obtains its neighbors' distance vectors
  - For each neighbor v, x obtains $\mathbf{D}_v = [D_v(y): y \in N]$
- Each node v periodically sends D_v to its neighbors
  - And neighbors update their own distance vectors
  - $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- Over time, the distance vector D_x converges

Distance Vector Example: Step 1

Optimum 1-hop paths

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

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

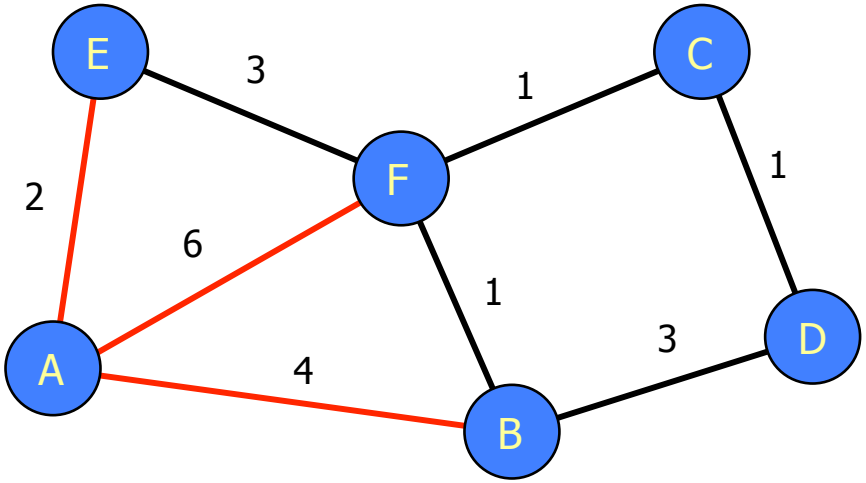


Table for C		
Dst	Cst	Hop
A	∞	–
B	∞	–
C	0	C
D	1	D
E	∞	–
F	1	F

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

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

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

Distance Vector Example: Step 2

Optimum 2-hop paths

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

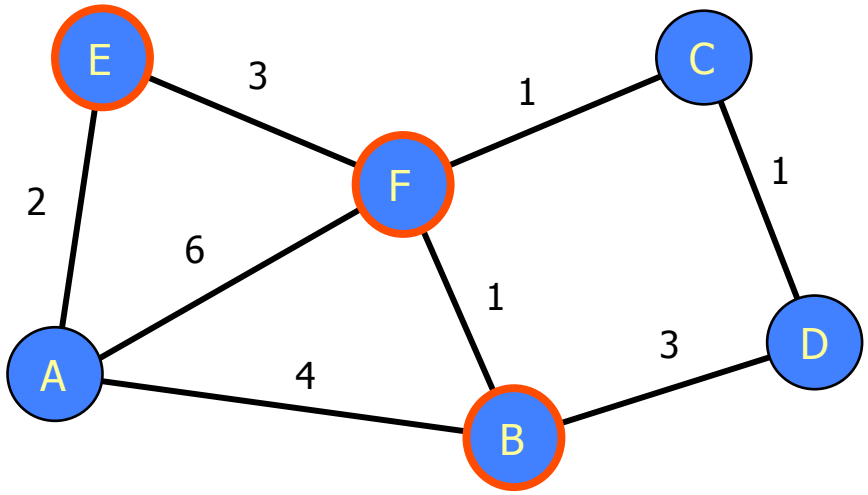


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	7	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	∞	–	D	2	C
E	4	F	E	∞	–	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Distance Vector Example: Step 3

Optimum 3-hop paths

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

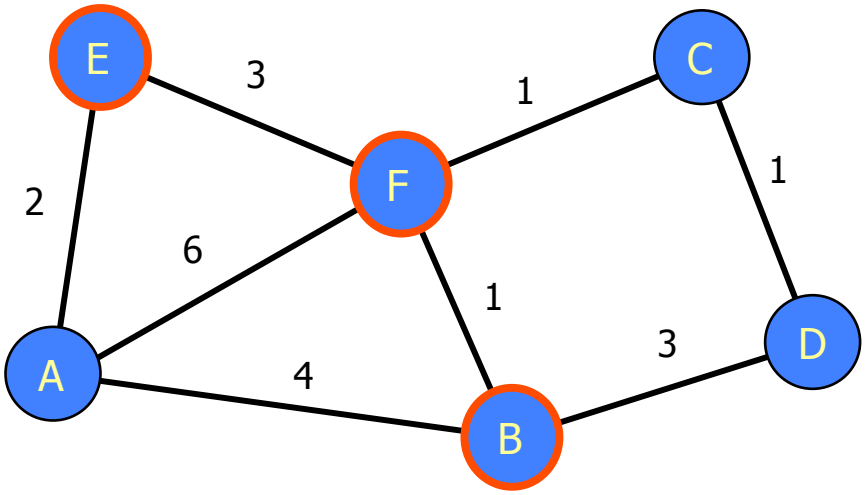
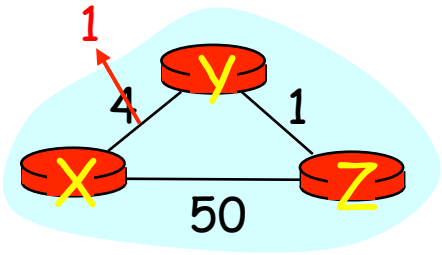


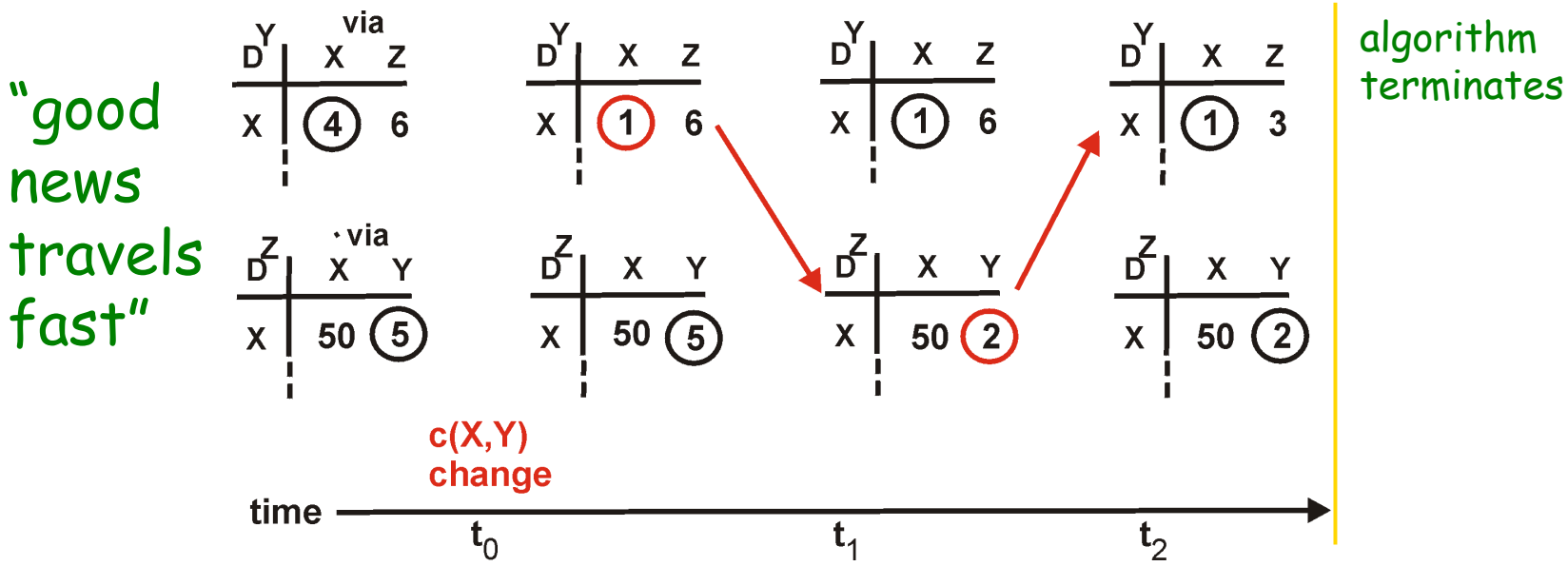
Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	5	F	D	2	C
E	4	F	E	5	C	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

DV: Good News Travels Fast

Link cost changes:

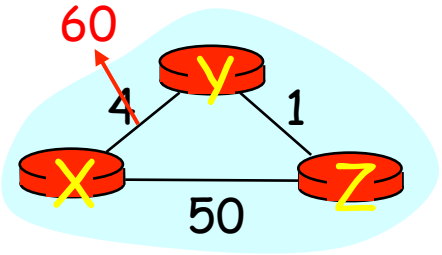
- Node detects local link cost change
- Updates the distance table
- If cost change in least cost path, notify neighbors

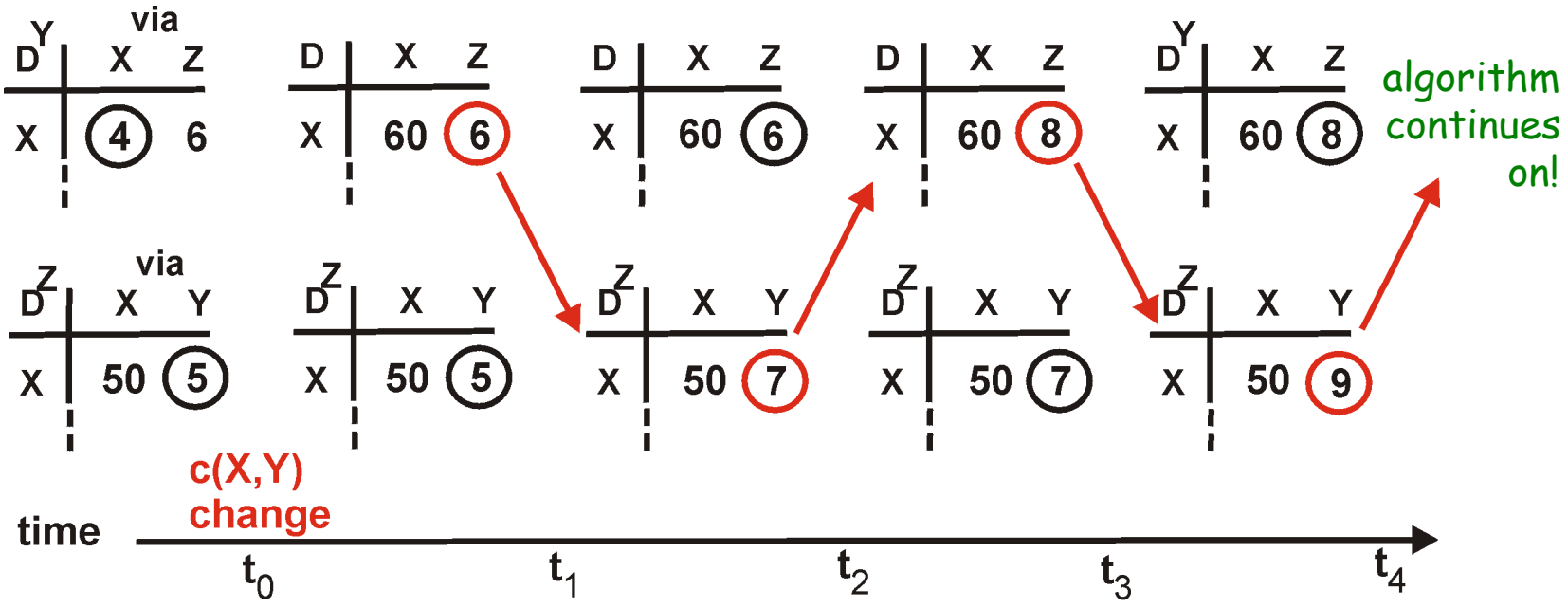




DV: Bad News Travels Slow

Count-To-Infinity Problem

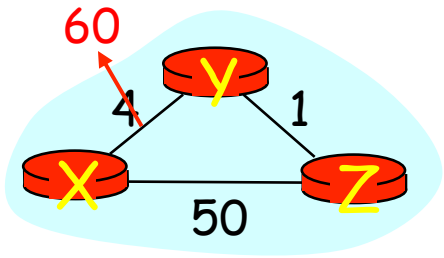


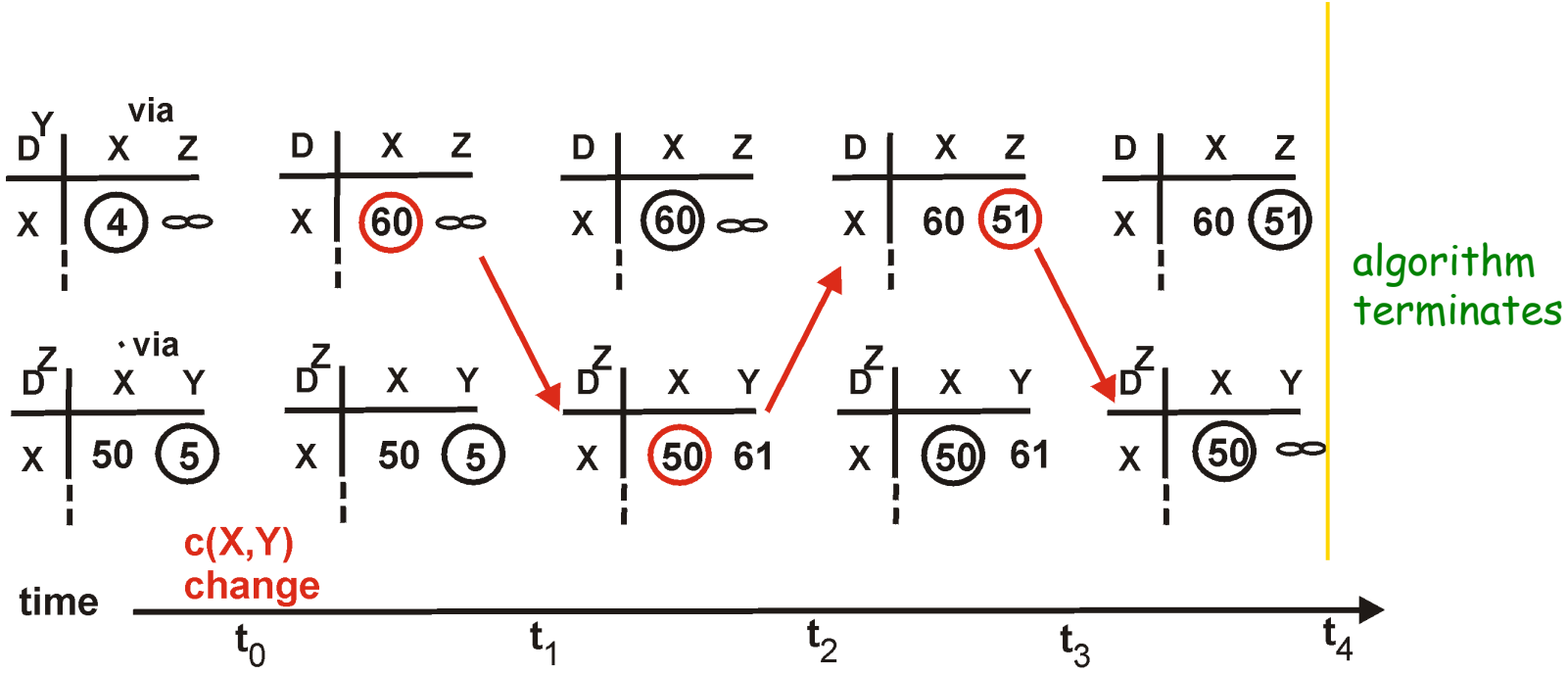


Distance Vector: Poison Reverse

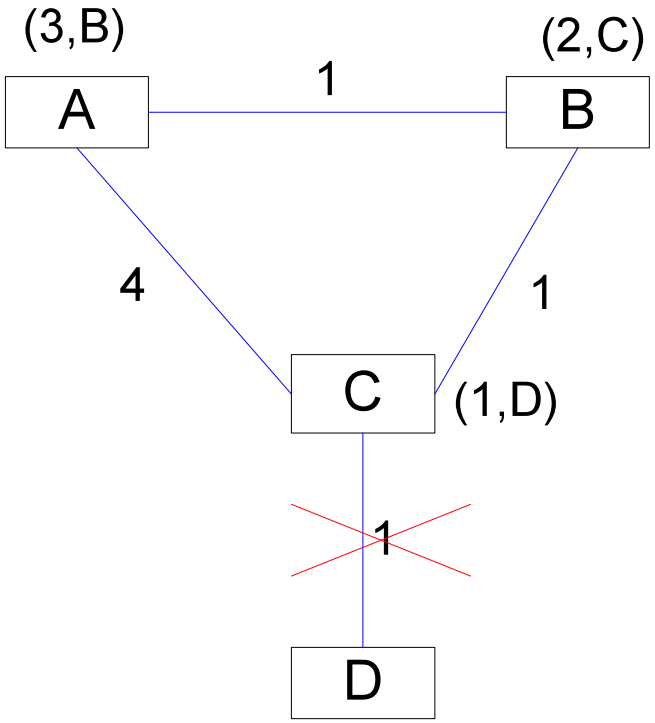
If Z learns route to X through Y

- Split horizon: Z does not advertise that route to Y
- Poison Reverse: Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)





DV: Poisoned Reverse Fails Too



A	B	C
(3,B)	(2,C)	(1,D)
(3,B)	∞	(7,A)
∞	(8,C)	(7,A)
(9,B)	(8,C)	∞
(9,B)	∞	(13,A)
∞	(14,C)	(13,A)
…	…	…

Routing Information Protocol (RIP)

- Distance vector protocol
  - Nodes send distance vectors every 30 seconds
  - … or, when an update causes a change in routing
- Link costs in RIP
  - All links have cost 1
  - Valid distances of 1 through 15
  - … with 16 representing infinity
  - Small "infinity" → smaller "counting to infinity" problem
- RIP is limited to fairly small networks

Link State vs. Distance Vector (1)

Message complexity

- LS: with n nodes, E links, O (nE) msgs sent each
- 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 have routing loops
  - count-to-infinity problem
  - Good news travels fast
  - Bad news travels slowly

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 through network

Link State vs. Distance Vector (2)

- *Message complexity*
  - LS requires *more* messages to be exchanged, DV requires *larger* messages
- *Speed of convergence*
  - DV can converge slower, but LS needs more overhead to flood messages
- *Robustness*
  - LS is somewhat more robust since each node calculate all the routes by itself, less dependent on a few routers' errors
- *Efficiency*
  - Hard to say, neither is a winner, both are used in practice