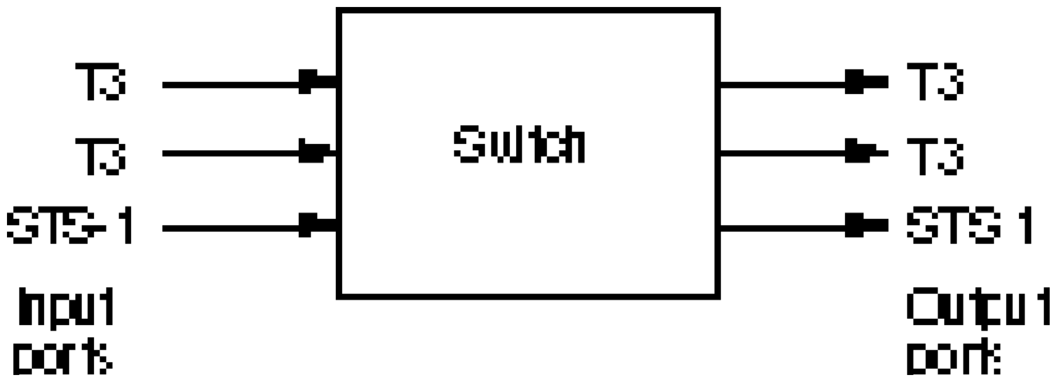
Switching, Forwarding and Routing

Scalable Networks

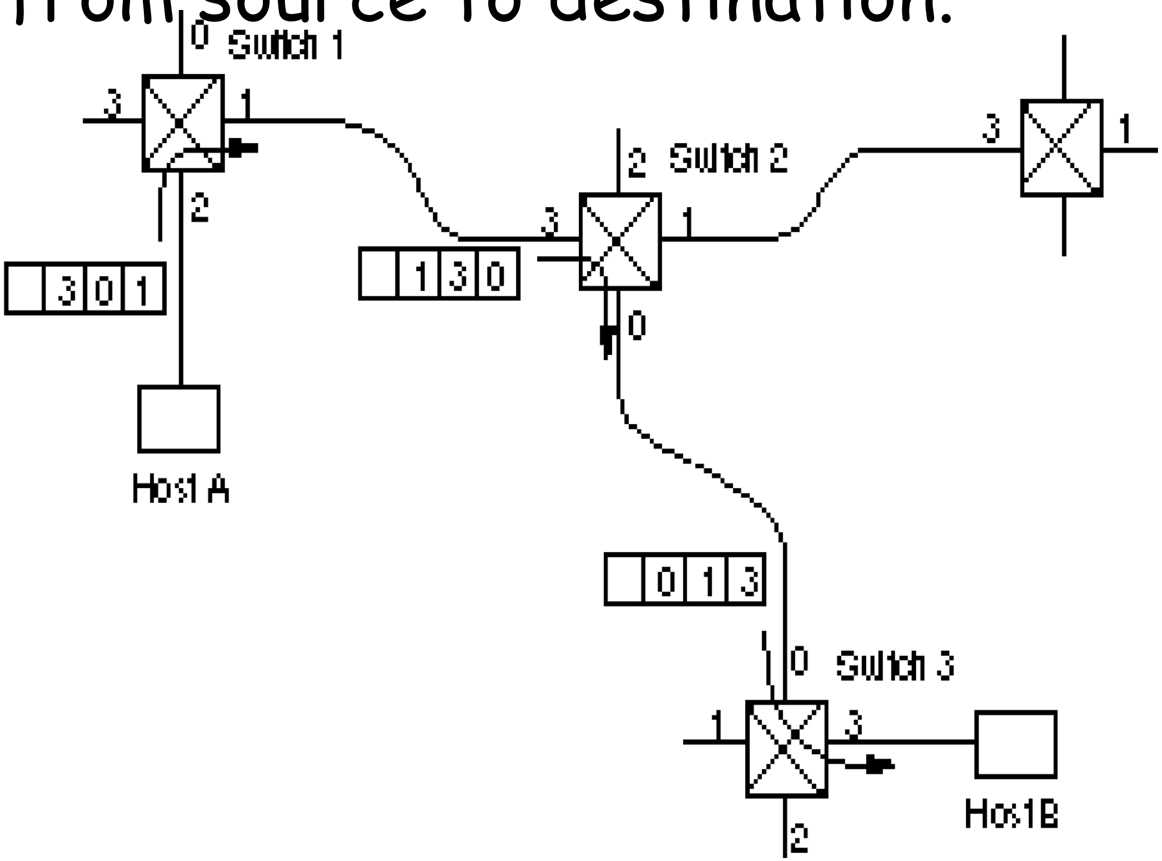
Switch: Forwards packets from input port to output port; port selected based on destination address in packet header.



- Can build networks that cover large geographic areas
- Can build networks that support large numbers of hosts
- Can add new hosts without affecting performance of existing hosts

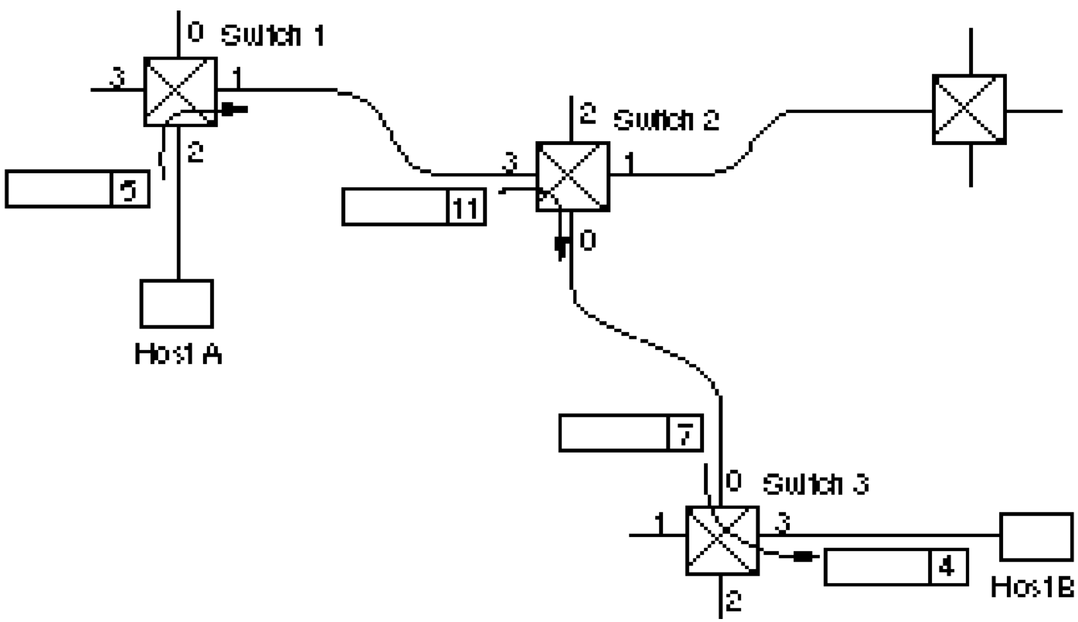
Source Routing

- Address contains sequence of ports on path from source to destination.



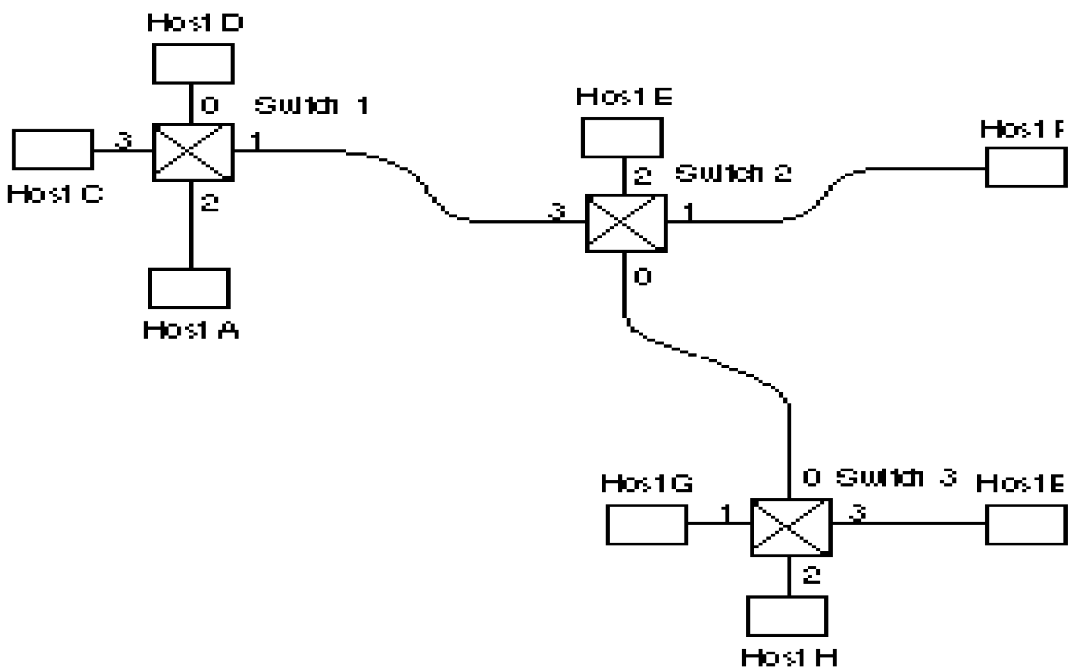
Virtual Circuit Switching

- Explicit connection setup (and tear-down) phase
- Subsequent packets follow same circuit
- Analogy: phone call
- Sometimes called *connection-oriented* model
- Each switch maintains a VC table.



Datagrams

- No connection setup phase
- Each packet forwarded independently
- Analogy: postal system
- Sometimes called *connectionless* model
- Each switch maintains a forwarding (routing) table



Virtual Circuit versus Datagram

Virtual Circuit Model:

- Typically wait full RTT for connection setup before sending first data packet.
- While the connection request contains the full address for destination, each data packet contains only a small identifier, making the per-packet header overhead small.
- If a switch or a link in a connection fails, the connection is broken and a new one needs to be established.
- Connection setup provides an opportunity to reserve resources.

Datagram Model:

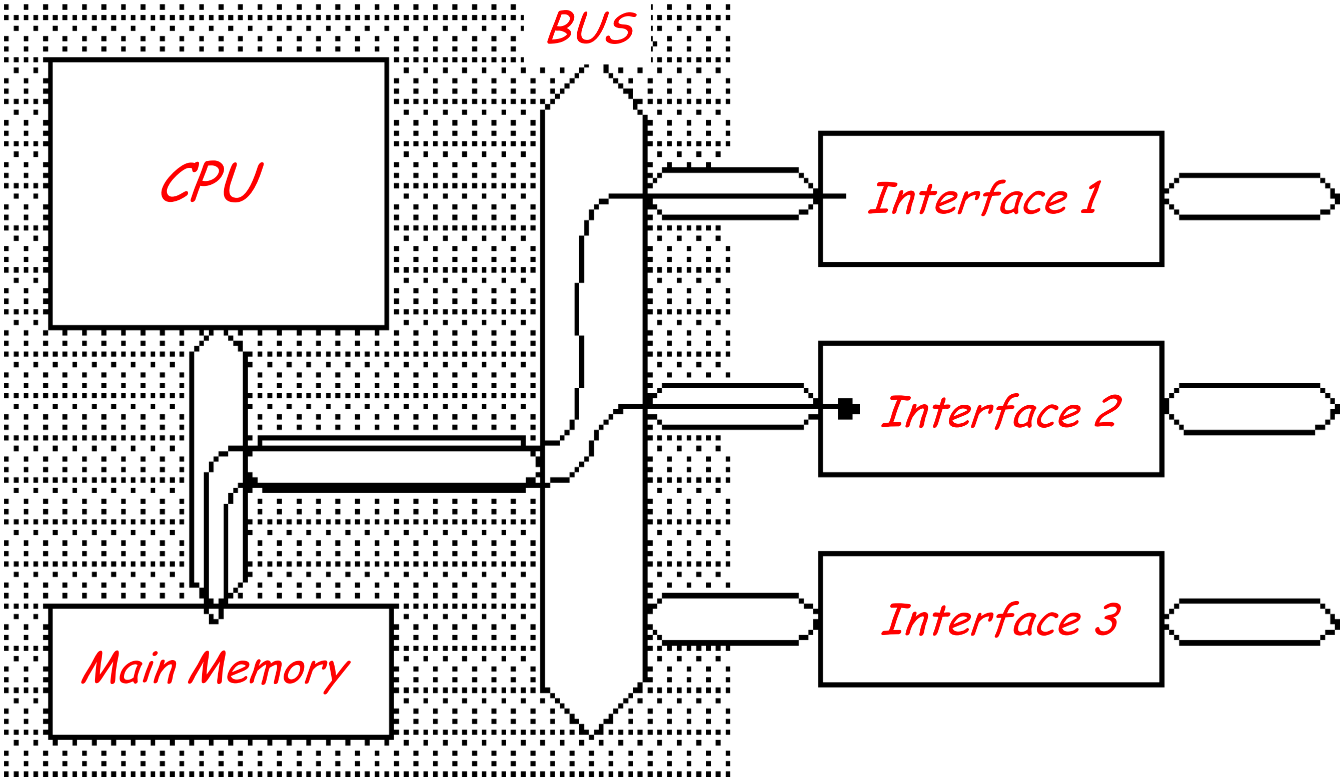
- There is no round trip time delay waiting for connection setup; a host can send data as soon as it is ready.
- Source host has no way of knowing if the network is capable of delivering a packet or if the destination host is even up.
- Since packets are treated independently, it is possible to route around link and node failures.
- Since every packet must carry the full address of the destination, the overhead per packet is higher than for the connection-oriented model.

The Problem of Congestion

- Congestion vs. Contention
  - Contention is the competition for an output link, or access to the medium.
  - Congestion occurs when too many packets are queued and the switch (or router) runs out of buffer space. Newly arriving packets may have to be dropped.
- X.25 (VCs) have defined away the problem of congestion! (WHY? HOW?)
- X.25 VCs are very conservative in terms of resource management.
- The Datagram Model takes an optimistic approach. (HOW?) What problems could occur?

Performance

Switches can be built from a general-purpose workstations; will consider special-purpose hardware later.



- Aggregate bandwidth
  - 1/2 of the I/O bus bandwidth
  - capacity is shared among all hosts connected to switch
  - example: 800Mbps bus can support 8 T3 ports
- Packets-per-second
  - must be able to switch small packets
  - 15,000 packets-per-second is an achievable number
  - example: 64-byte packets implies 7.69Mbps

Routing Strategies

Routing

- The primary function of a packet network is to accept packets from a source and deliver them to a destination node.
- The process of forwarding the packets through the network is referred to a routing.
- Routing mechanisms have a set of requirements:
  - correctness
  - simplicity
  - robustness
  - stability
  - fairness

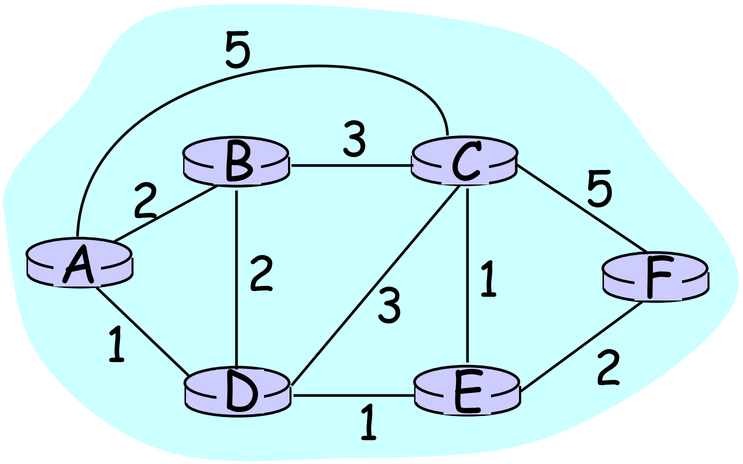
- Most important:
  - optimality
  - efficiency
- Routing directly impacts the performance of the network! WHY?
- In order to route packets on optimal routes through the network to their destinations, we must first decide what is to be optimized:
  - delay
  - cost
  - throughput

- Routing decisions are generally based on some knowledge of the state of the network.
  - Delay on certain links
  - Cost through certain nodes
  - Packet loss
  - etc.
- This information may have to be dynamically collected. This leads to overhead which in turn reduces the utilization.

Routing

Routing protocol

Goal: determine "good" path (sequence of routers) thru network from source to dest.



Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level

- "good" path:
  - typically means minimum cost path
  - other def's possible

Routing Algorithm classification

Global or decentralized information?

Global:

- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic?

Static:

- routes change slowly over time

Dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Different Types of Routing

- Fixed Routing:
  - Static Routing Tables, Pre-computed Routes
- Flooding:
  - Simple but inefficient! WHY?
- Hot Potato Routing
  - Simple, not very efficient, unpredictable
- Random Routing
  - Simple, unpredictable, statistically fair (locally)
- Adaptive Routing
  - sophisticated, expensive, efficient, complex...

Random Routing

- Sometimes called *probabilistic routing*!
- Here, the probability of a packet being forwarded on a particular link is a function of conditions on this link.

$$P_i = \frac{R_i}{\sum_j R_j}$$

  - P_i = Probability of link i being selected
  - R_i = Data rate on link i

- Note: Random Routing is probabilistic, i.e., the link with the largest capacity may not be the one chosen for every transmission.
- We can formulate a static and dynamic (adaptive) version of the routing algorithm.

- Can you think of other measurements (metrics) to compute P_i ?

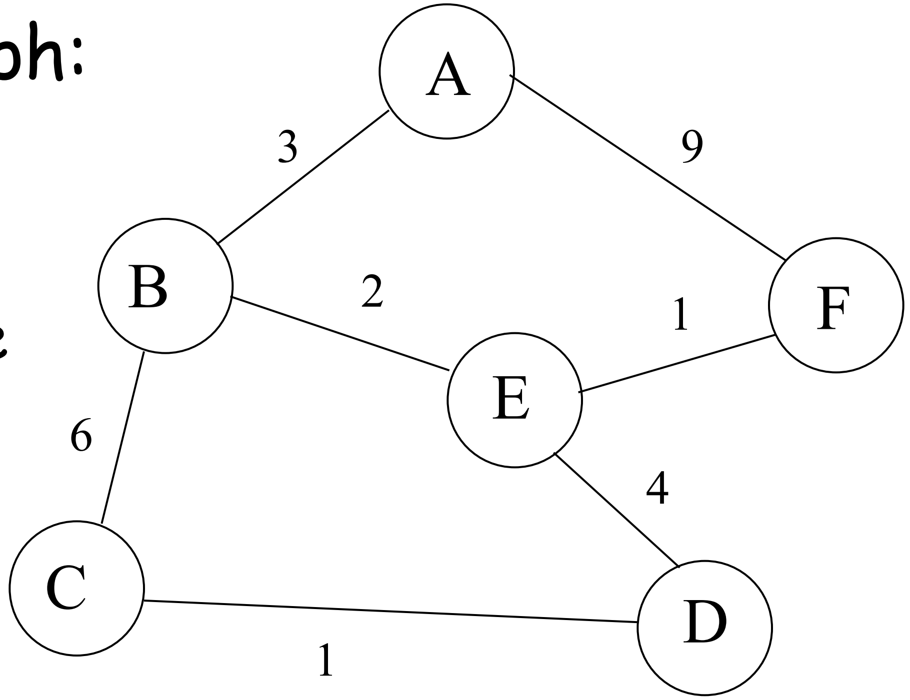
Adaptive Routing

- Adaptive Routing Techniques are used in almost all packet-switching networks.
  - ARPANET
- Routing decisions change in response to changes in the network.
  - Network Failure
  - Congestion
- Adaptive routing strategies can improve performance.
- Adaptive routing strategies can aid congestion control.

- Adaptive routing mechanisms are based on shortest path algorithm usually developed in the field of graph theory.

- The trick is to formulate the centralized form of these algorithm to work in a distributed setting, such as a communication network.
- The information upon routing decisions are based may come from
  - local measurements
  - adjacent nodes
  - all nodes in the network

- Problem:
  - Find a least cost path between any two nodes.
- Network as a graph:
  - Vertices
  - Edges
  - Cost on each edge



- Some of the shortest-path algorithms established in traditional graph theory are:
  - Dijkstra's shortest path algorithm
  - Bellman-Ford Algorithm
  - Floyd-Warshall Algorithm
- The main difference between the algorithms is the type of augmentation through each iteration.
  - Dijkstra: nodes
  - Bellman-Ford: number of arcs (links) in the path
  - Floyd-Warshall: set of nodes in the path (all s-d pairs)
- These algorithms have been formulated in a centralized manner and must be mapped into a distributed environment.

A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

Notation:

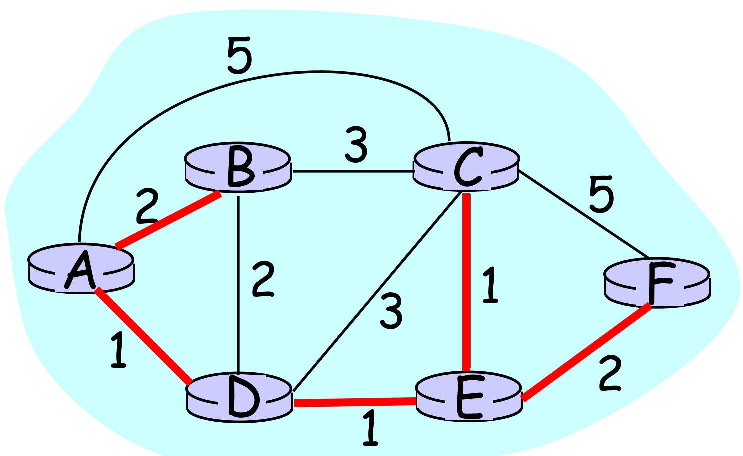
- c(i,j): link cost from node i to j. cost infinite if not direct neighbors
- D(v): current value of cost of path from source to dest. V
- p(v): predecessor node along path from source to v, that is next v
- N: set of nodes whose least cost path definitively known

Dijsktra's Algorithm

```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infty

Loop
  find w not in N such that D(w) is a minimum
  add w to N
  update D(v) for all v adjacent to w and not in N:
     D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N
```

Dijkstra's algorithm: example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



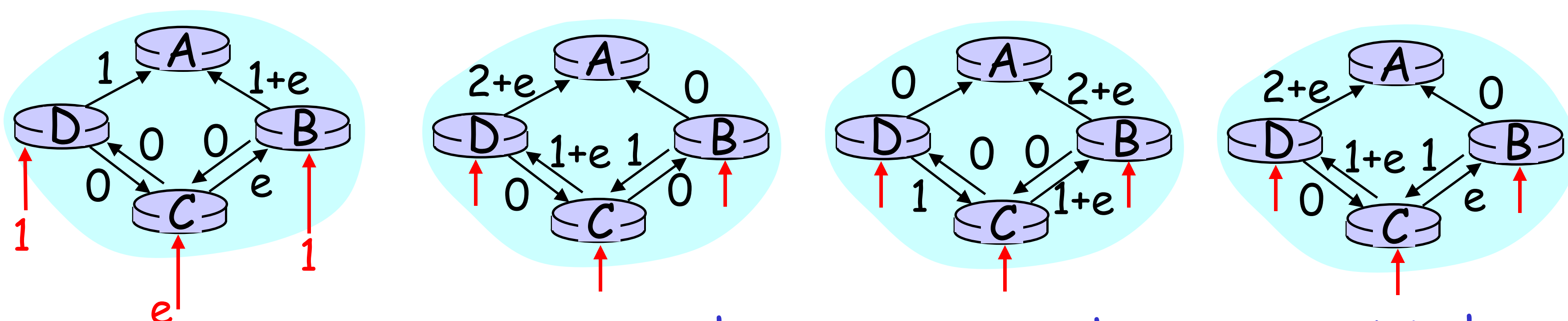
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- n*(n+1)/2 comparisons: O(n**2)
- more efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., link cost = amount of carried traffic



initially … recompute routing … recompute … recompute

Bellman-Ford (Distance Vector)

- The algorithm iterates on # of arcs in a path.
- The original algorithm is a single destination shortest path algorithm.
- Let $D^{(h)}_i$ be the shortest ($\leq$ h) path length from node i to node 1 (the destination).
- By convention, $D^{(h)}_1 = 0 \; \forall h$.
- Assumptions:
  - There exists at least one path from every node to the destination
  - All cycles not containing the destination have nonnegative length (cost).

- NOTE: Let $SD(i,j)$ be the shortest distance from node i to node j. In an undirected graph, we clearly have: $SD(i,j) = SD(j,i)$.
- This may not be true for a *Digraph.*

- Why is the assumption of cycles with nonnegative cost important?

- Length (hops) is just one of many possible routing metrics. Can you think of others?

- The Bellman-Ford Algorithm:
  - Step 1: Set $D^{(0)}_i = \infty \ \forall i$
  - Step 2: For each $h \geq 0$ compute $D^{(h+1)}_i$ as

$$D^{(h+1)}_i = \min_j [D^{(h)}_j + d_{j,i}] \quad \forall i \neq 1$$

  - where $d_{j,i}$ is the cost (length) of link $l_{j,i}$

- We say that the algorithm has terminated when $D^{(h)}_i = D^{(h-1)}_i \ \forall i$
- In a network with N nodes, the algorithm terminates after at most N iterations!

Distance Vector Routing Algorithm

iterative:

- continues until no nodes exchange info.
- *self-terminating*: no "signal" to stop

asynchronous:

- nodes need *not* exchange info/iterate in lock step!

distributed:

- each node communicates *only* with directly-attached neighbors

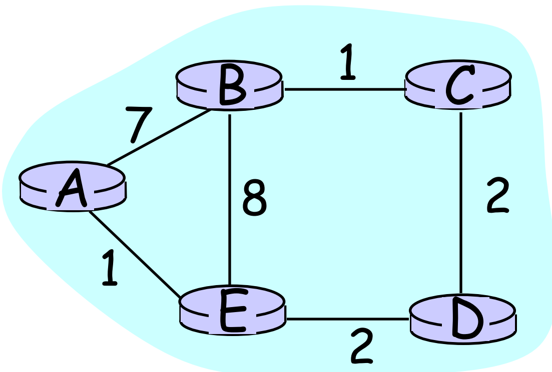
Distance Table data structure

- each node has its own
- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

$$D^X(Y,Z) = \text{distance } \textit{from } X \textit{ to } Y, \textit{ via } Z \text{ as next hop}$$

$$= c(X,Z) + \min_w \{D^Z(Y,w)\}$$

Distance Table: example



$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

(columns: cost to destination via; rows: destination)

$$D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\}$$
$$= 2+2 = 4$$

$$D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\}$$
$$= 2+3 = 5$$

loop!

$$D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\}$$
$$= 8+6 = 14$$

loop!

Distance table gives routing table

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

	Outgoing link to use, cost
A	A,1
B	D,5
C	D,4
D	D,4

destination

Distance table → Routing table

Distance Vector Routing: overview

Iterative, asynchronous:
each local iteration caused by:

- local link cost change
- message from neighbor: its least cost path change from neighbor

Distributed:

- each node notifies neighbors *only* when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary

Each node:

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

recompute distance table

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

Distance Vector Algorithm:

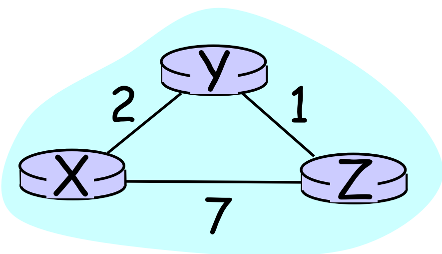
At all nodes, X:

Initialization:
  for all adjacent nodes v:
    $D^X(*,v)$ = infty   /* the * operator means "for all rows" */
    $D^X(v,v) = c(X,v)$
  for all destinations, y
    send $\min_w D^X(y,w)$ to each neighbor /* w over all X's neighbors */

Distance Vector Algorithm (cont.):

```
loop
  wait (until I see a link cost change to neighbor V
        or until I receive update from neighbor V)

  if (c(X,V) changes by d)
    /* change cost to all dest's via neighbor v by d */
    /* note: d could be positive or negative */
    for all destinations y:  D^X(y,V) =  D^X(y,V) + d

  else if (update received from V wrt destination Y)
    /* shortest path from V to some Y has changed  */
    /* V has sent a new value for its  min_w DV(Y,w) */
    /* call this received new value is "newval"     */
    for the single destination y: D^X(Y,V) = c(X,V) + newval

  if we have a new min_w D^X(Y,w)for any destination Y
     send new value of min_w D^X(Y,w) to all neighbors

forever
```

Distance Vector Algorithm: example



D^X	cost via Y	Z
dest Y	2	∞
dest Z	∞	7

D^Y	cost via X	Z
dest X	2	∞
dest Z	∞	1

D^Z	cost via X	Y
dest X	7	∞
dest Y	∞	1

D^X	cost via Y	Z
dest Y	2	8
dest Z	3	7

D^Y	cost via X	Z
dest X	2	8
dest Z	9	1

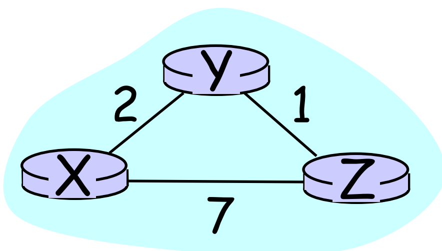
D^Z	cost via X	Y
dest X	7	3
dest Y	9	1

D^X	cost via Y	Z
dest Y		
dest Z		

D^Y	cost via X	Z
dest X		
dest Z		

D^Z	cost via X	Y
dest X		
dest Y		

Distance Vector Algorithm: example



D^X (dest)	cost via Y	cost via Z
Y	2	∞
Z	∞	7

D^Y (dest)	cost via X	cost via Z
X	2	∞
Z	∞	1

D^Z (dest)	cost via X	cost via Y
X	7	∞
Y	∞	1

D^X (dest)	cost via Y	cost via Z
Y	2	8
Z	3	7

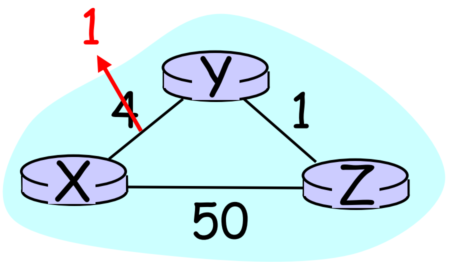
$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\} = 7+1 = 8$$

$$D^X(Z,Y) = c(X,Y) + \min_w\{D^Y(Z,w)\} = 2+1 = 3$$

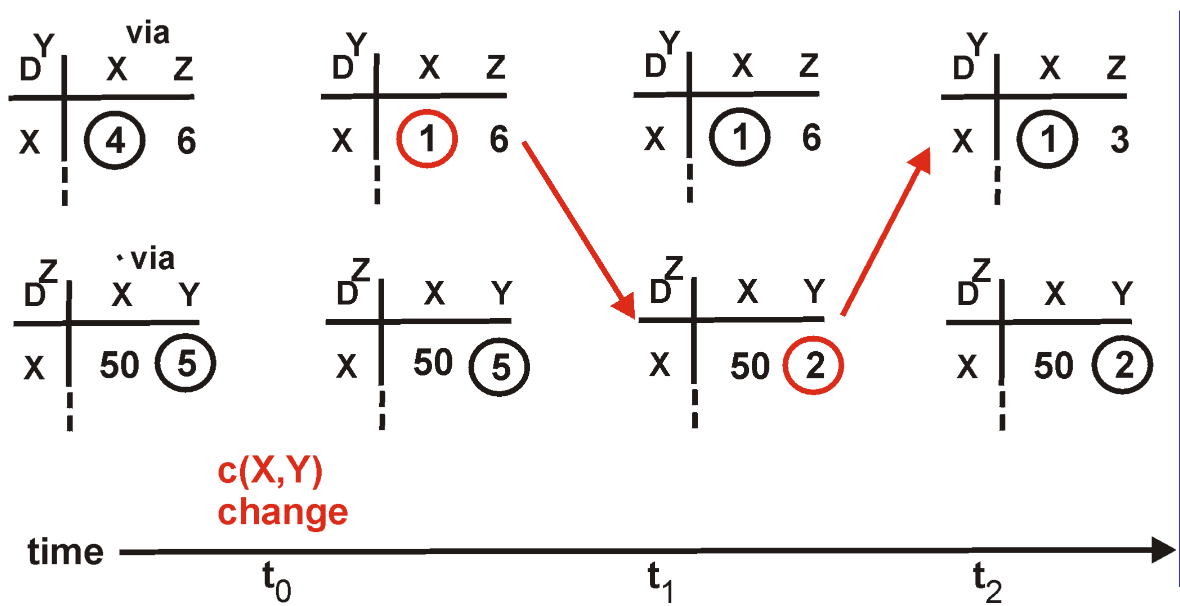
Distance Vector: link cost changes

Link cost changes:

- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23,24)



"good news travels fast"

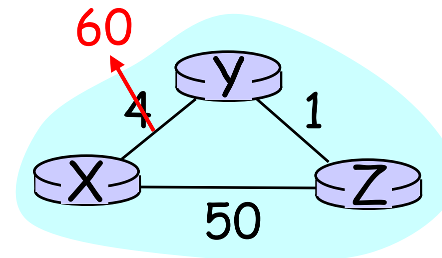


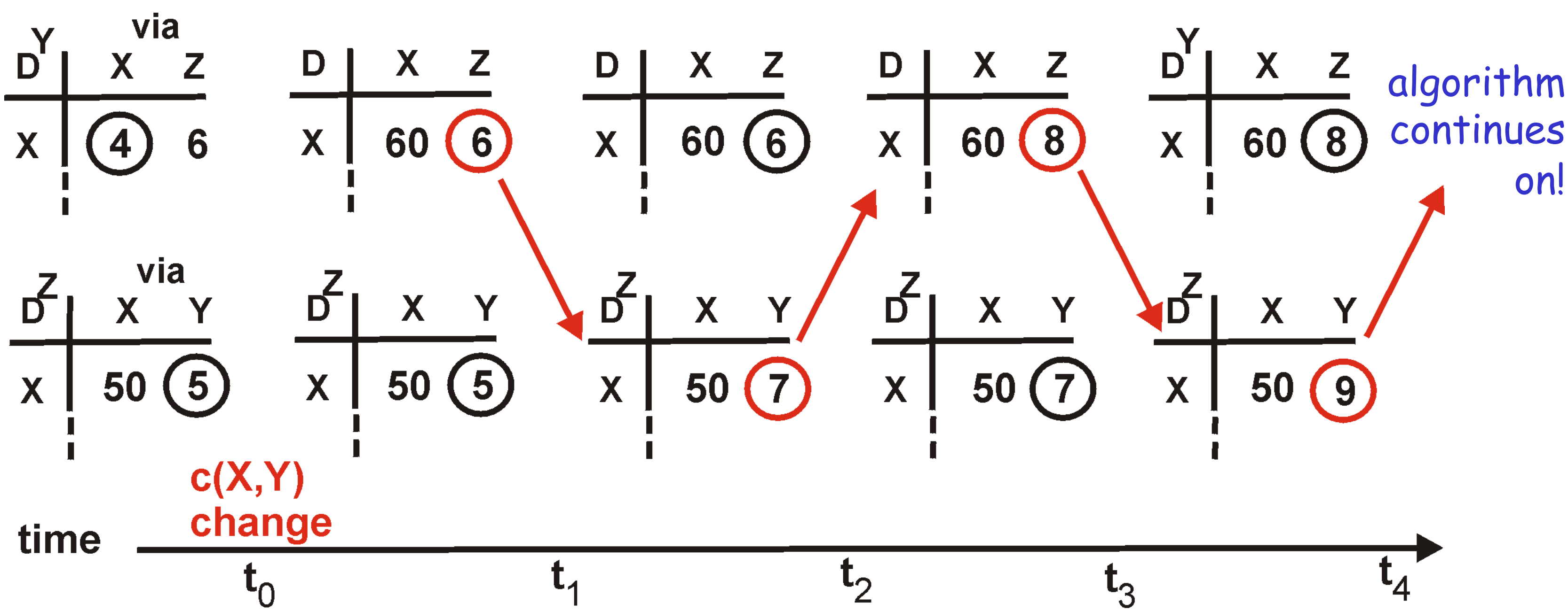
algorithm terminates

Distance Vector: link cost changes

Link cost changes:

- good news travels fast
- bad news travels slow - "count to infinity" problem!

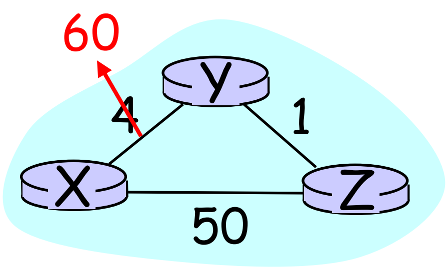


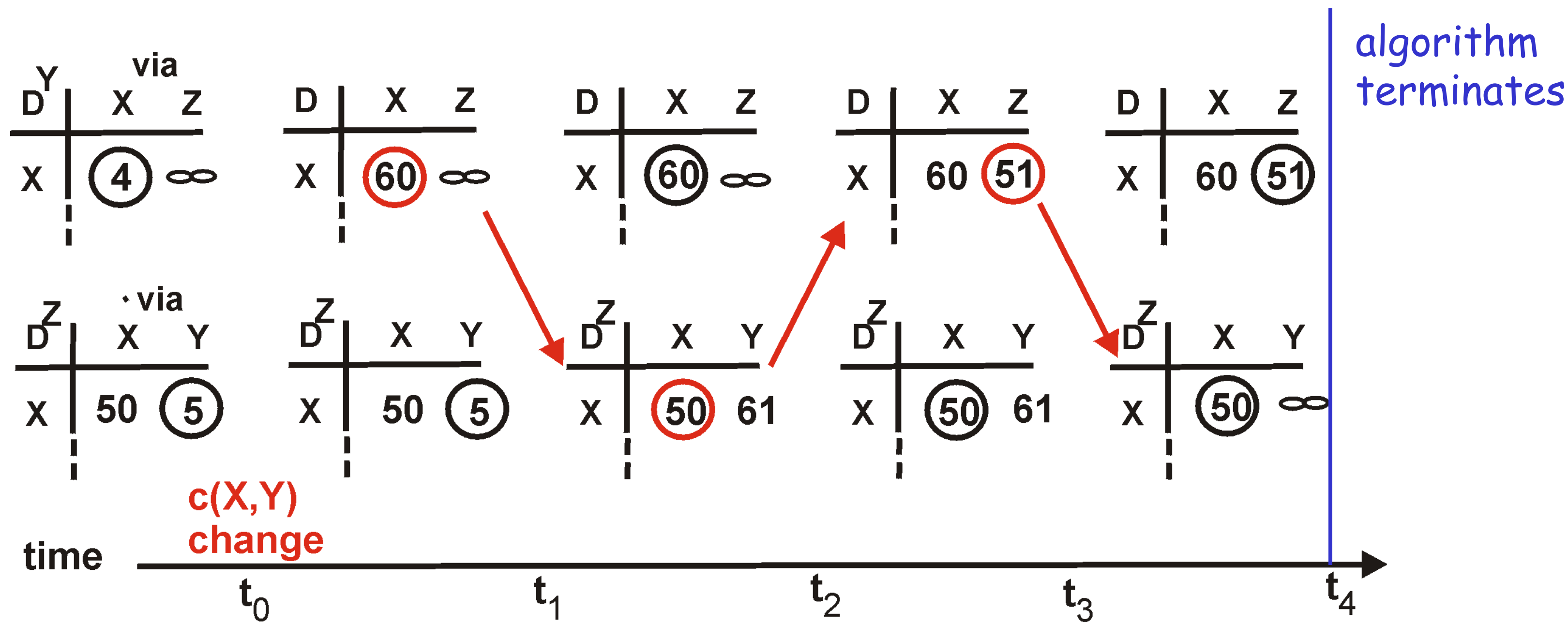


Distance Vector: poisoned reverse

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?





Comparison of LS and DV algorithms

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 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