Computer Networks COL 334/672

Network Layer

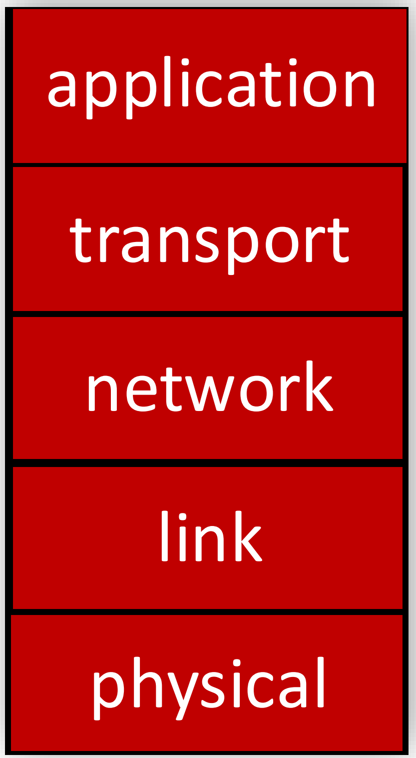
Tarun Mangla

Slides adapted from KR

Sem 1, 2024-25

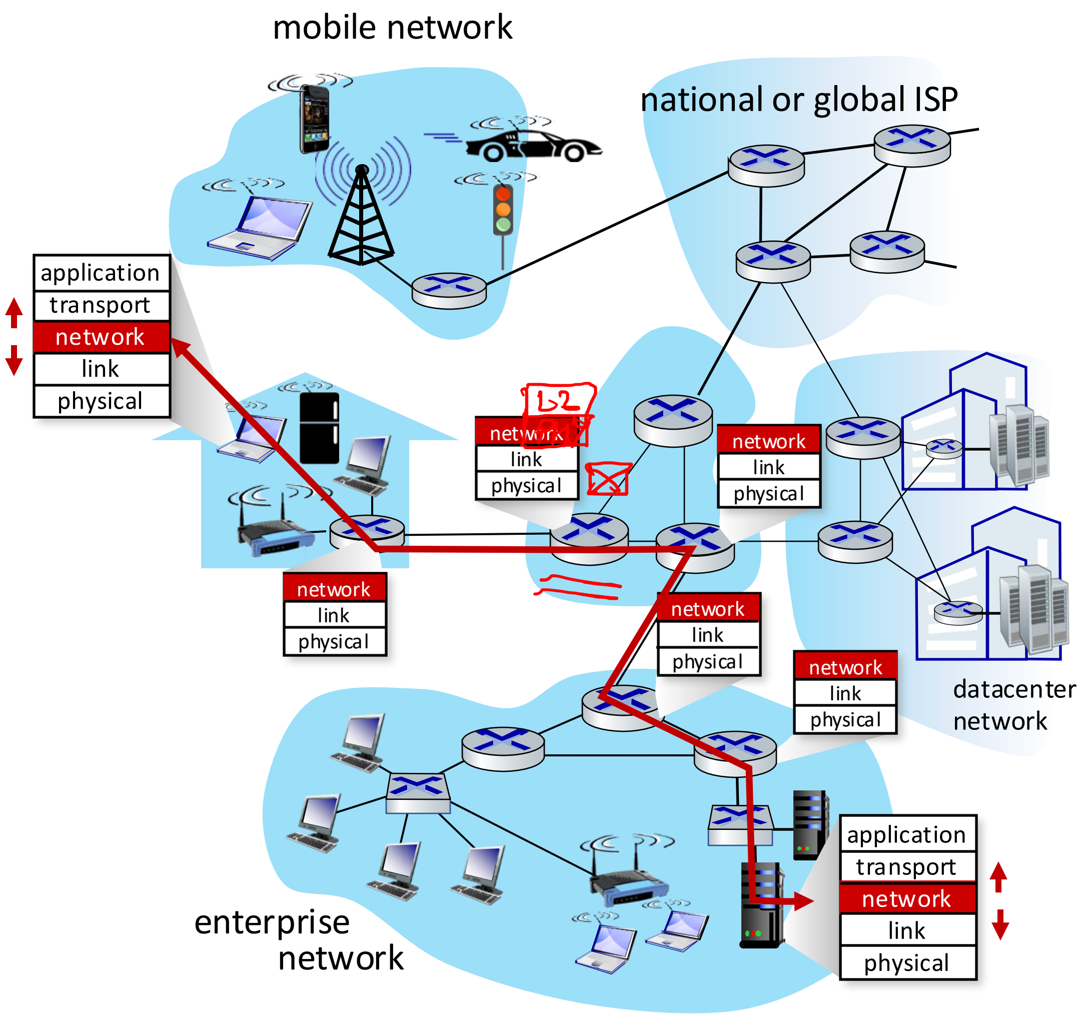
Layered Internet protocol stack

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"



Network-layer services and protocols

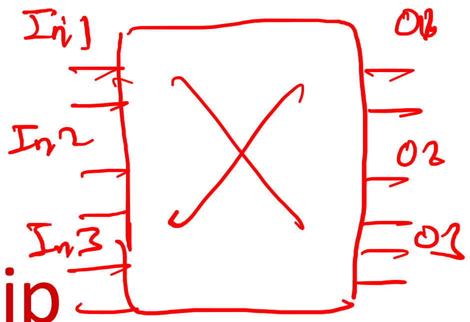
- transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path



Two key network-layer functions

forwarding

HS or ns



h/w

network-layer functions:

- *forwarding:* move packets from a router's input link to appropriate router output link

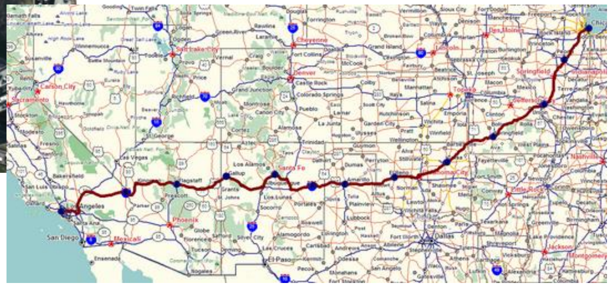
software

- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

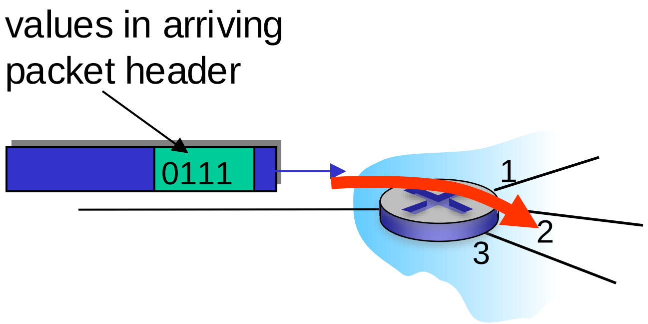

forwarding


routing

Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port



Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host

Routing Protocol Overview

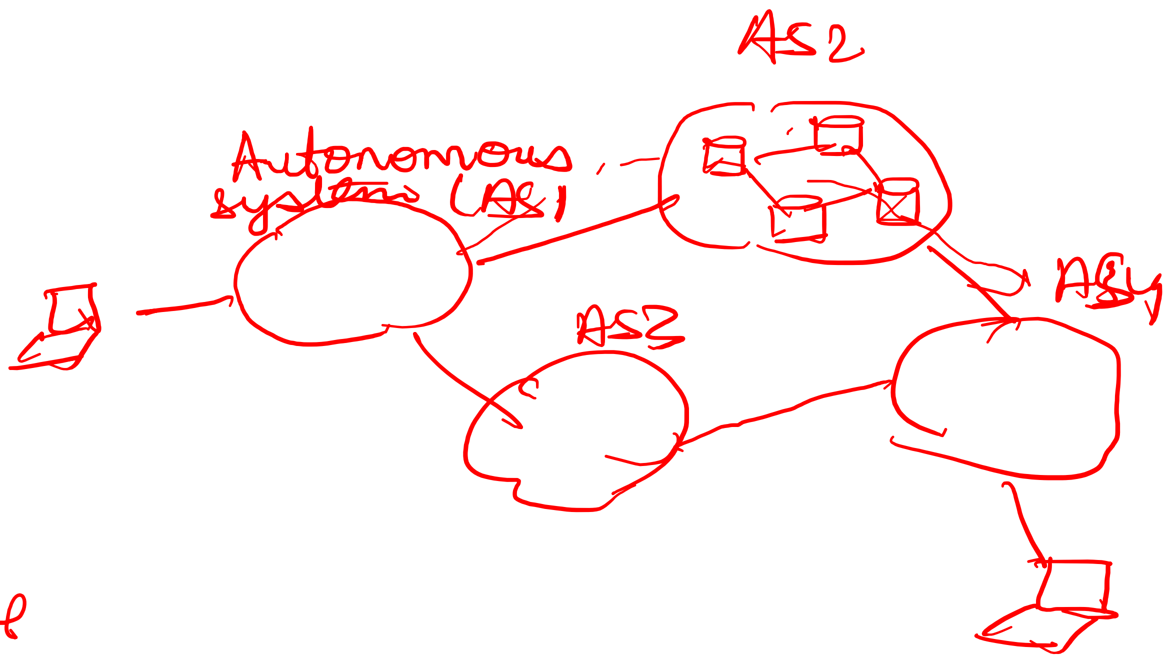
least latency, high throughput, least lossy / most reliable

→ secure path → least expensive

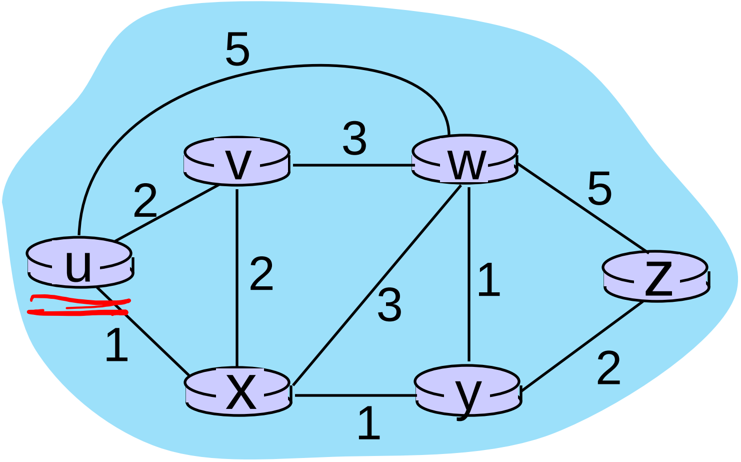
- **Goal:** determine "good" paths from sending host to receiving host through networks of routers
- **Good:** least congested, lowest latency, least cost
- **At what level?** BGP → IP
  - Intra-domain or Inter-domain
  -AS
- **How to do it?**
  - Centralized or distributed manner

  SDN or Software defined Network

  Traditional
- Provides scalability to the Internet



Intra-domain Routing: Graph Abstraction



$c_{a,b}$: cost of *direct* link connecting *a* and *b*

e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: $G = (N,E)$

N: set of routers = { *u, v, w, x, y, z* }

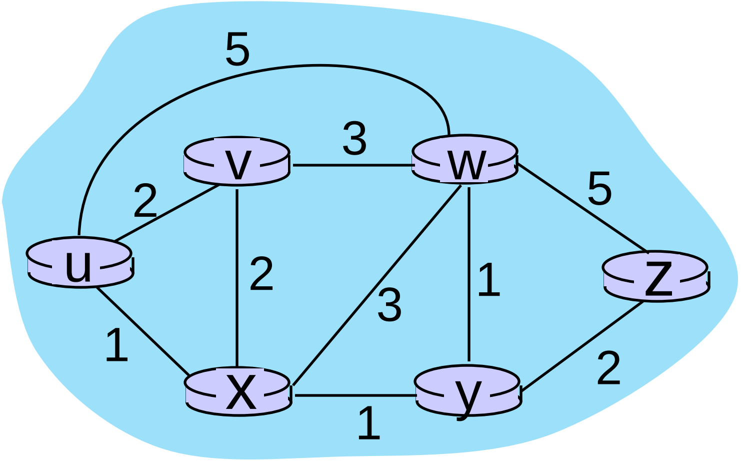
E: set of links ={ (*u,v*), (*u,x*), (*v,x*), (*v,w*), (*x,w*), (*x,y*), (*w,y*), (*w,z*), (*y,z*) }

How to determine shortest path from one node to all other nodes in a graph?

Dijkstra Algorithm

T' : $V - \{u\}$ → Pick the minimum distance vertex

T : $\{u\}$



Each node finds shortest path tree to all other nodes in the network

T' : unvisited node

T : visited node

$D(v)$, $D(x)$ . . .

$D(v) = \boxed{\min\{D(v),\ D(w) + c(w,v)\}}$

Dijkstra's link-state routing algorithm

- centralized: network 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 *forwarding table* for that node
- iterative: after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v)*: *current* estimate of cost of least-cost-path from source to destination *v*
- *p(v)*: predecessor node along path from source to *v*
- *N'*: set of nodes whose least-cost-path *definitively* known

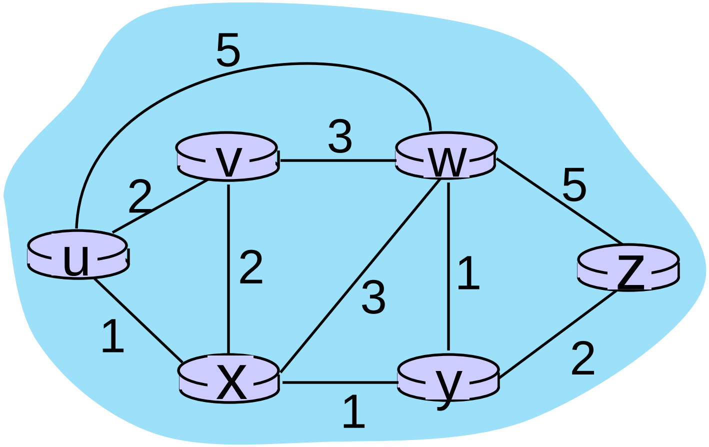
Dijkstra's link-state routing algorithm

Initialization:
  $N' = \{u\}$ /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */
      then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */
    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 least-path-cost to v is either old least-cost-path to v or known
  least-cost-path to w plus direct-cost from w to v */
until all nodes in N'

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	(2, u)	(5, u)	(1, u)	∞, -	∞, -
1						
2						
3						
4						
5						



Initialization:

$N' = \{u\}$

for all nodes v

  if v adjacent to u

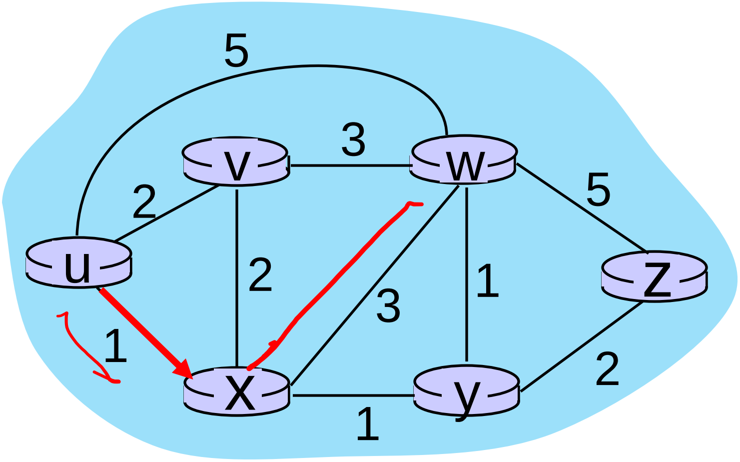
    then $D(v) = c_{u,v}$

  else $D(v) = \infty$

$D(v) = \min\{D(w), D(u) + C(u,v)\}$

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxv		4,x		2,x	∞
3	uxvy		3,y			4,y
4	uxvyw					4,y
5	uxvywz					



8 *Loop*

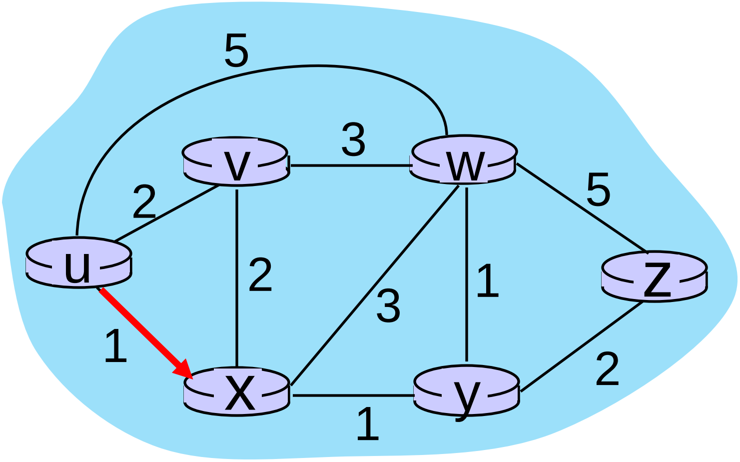
9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

Dest	Nexthop	Cost

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2						
3						
4						
5						



8 *Loop*

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

11 update $D(b)$ for all b adjacent to a and not in N' :

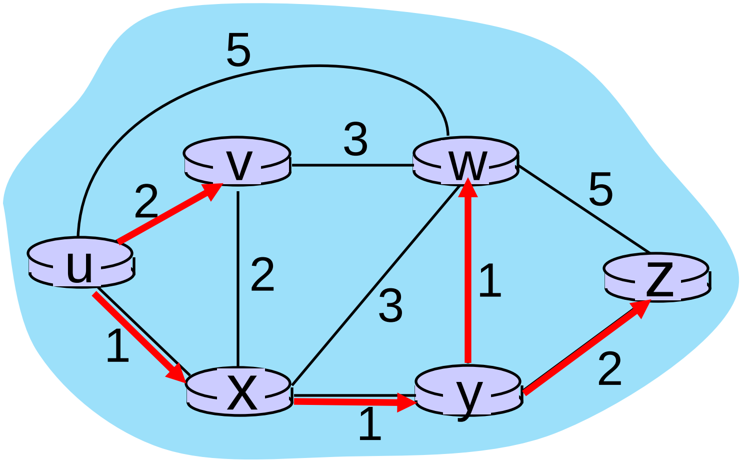
$$\mathbf{D(b) = \min (D(b), D(a) + c_{a,b})}$$

$D(v) = \min (D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$

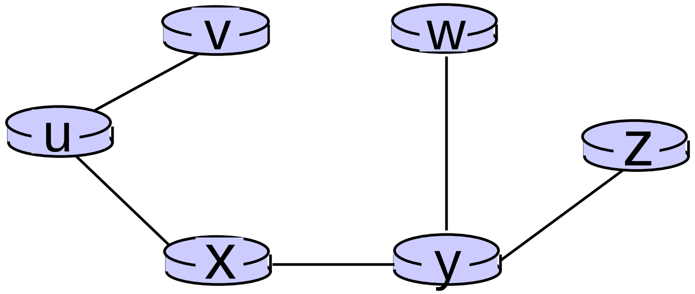
$D(w) = \min (D(w), D(x) + c_{x,w}) = \min (5, 1+3) = 4$ NEW!

$D(y) = \min (D(y), D(x) + c_{x,y}) = \min(inf,1+1) = 2$ NEW!

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



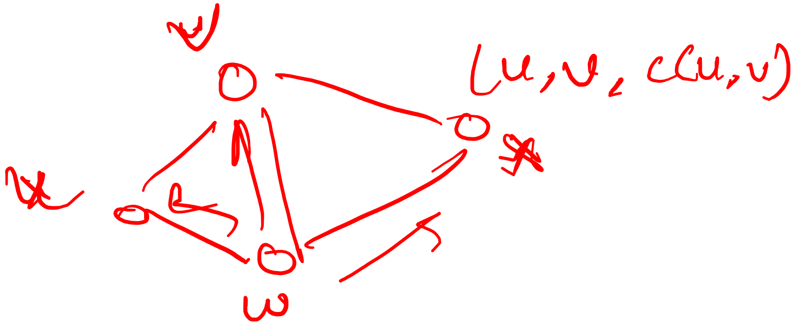
resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
~~x~~ z	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra's algorithm: discussion



algorithm complexity: *n* nodes

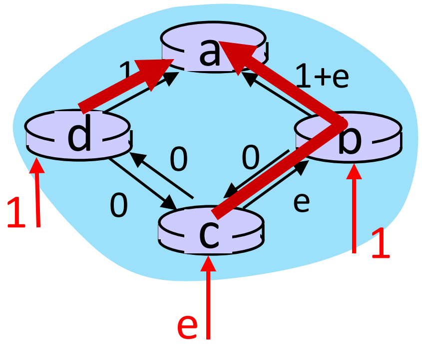
- each of *n* iteration: need to check all nodes, *w*, not in *N*
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n\log n)$

message complexity: link state announcement

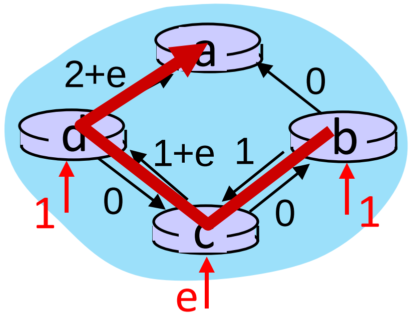
- each router must *broadcast* its link state information to other *n* routers
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

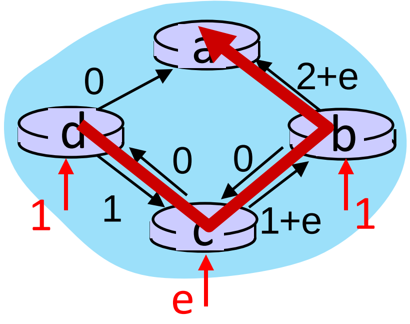
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



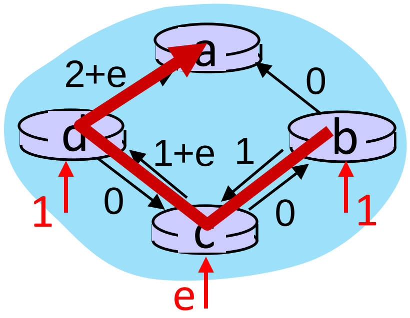
initially



given these costs, find new routing.... resulting in new costs



given these costs, find new routing.... resulting in new costs



given these costs, find new routing.... resulting in new costs

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

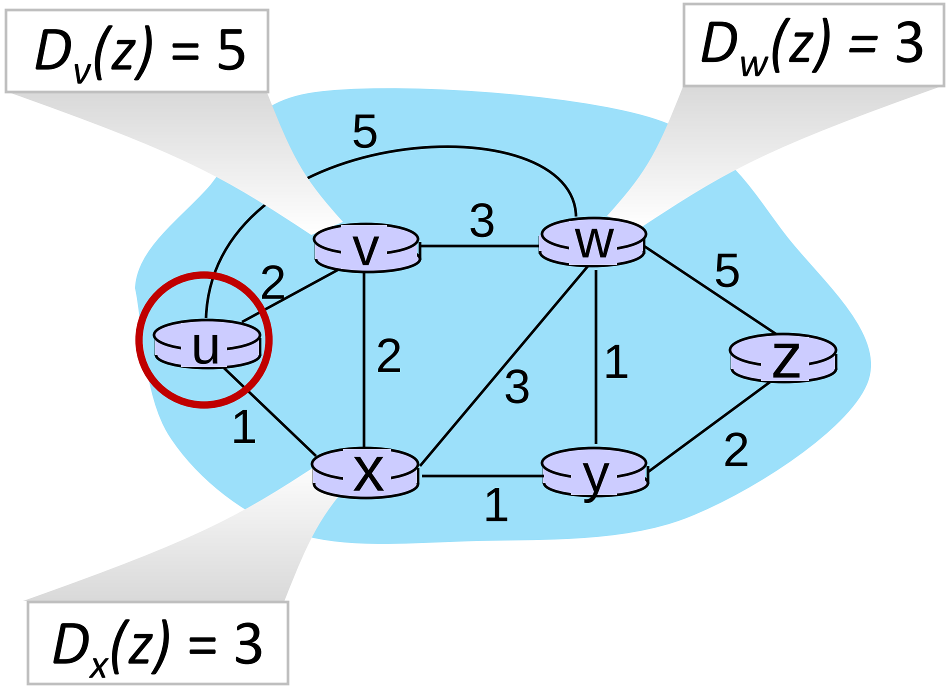
v's estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w,* know that for destination *z*:



D

Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z), \\ c_{u,x} + D_x(z), \\ c_{u,w} + D_w(z) \}$$

$$= \min \{2 + 5, \\ 1 + 3, \\ 5 + 3\} = 4$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Distance vector algorithm

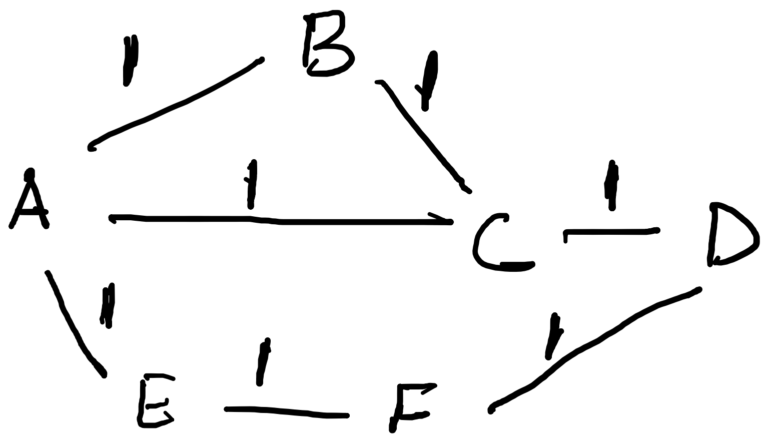
key idea:

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

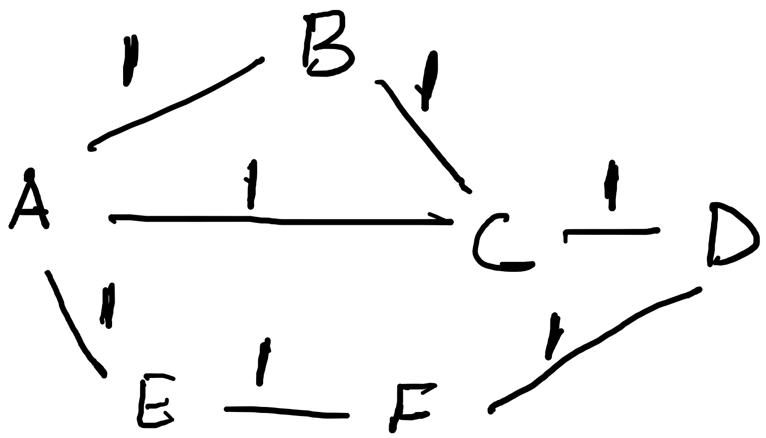
$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\} \quad \text{for each node } y \in N$$

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

Distance vector: Example



Distance vector: Example



Distance vector algorithm:

each node:

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

↓

recompute DV estimates using DV received from neighbor

↓

if DV to any destination has changed, *notify* neighbors

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!