Network Layer: Routing Protocols

A/PROF. DUY NGO

Learning Objectives

5.1 introduction

5.2 routing protocols

- link state
- distance vector

Network-Layer Functions

Recall: two network-layer functions:

- **forwarding:** move packets from router's input to appropriate router output — **data plane**

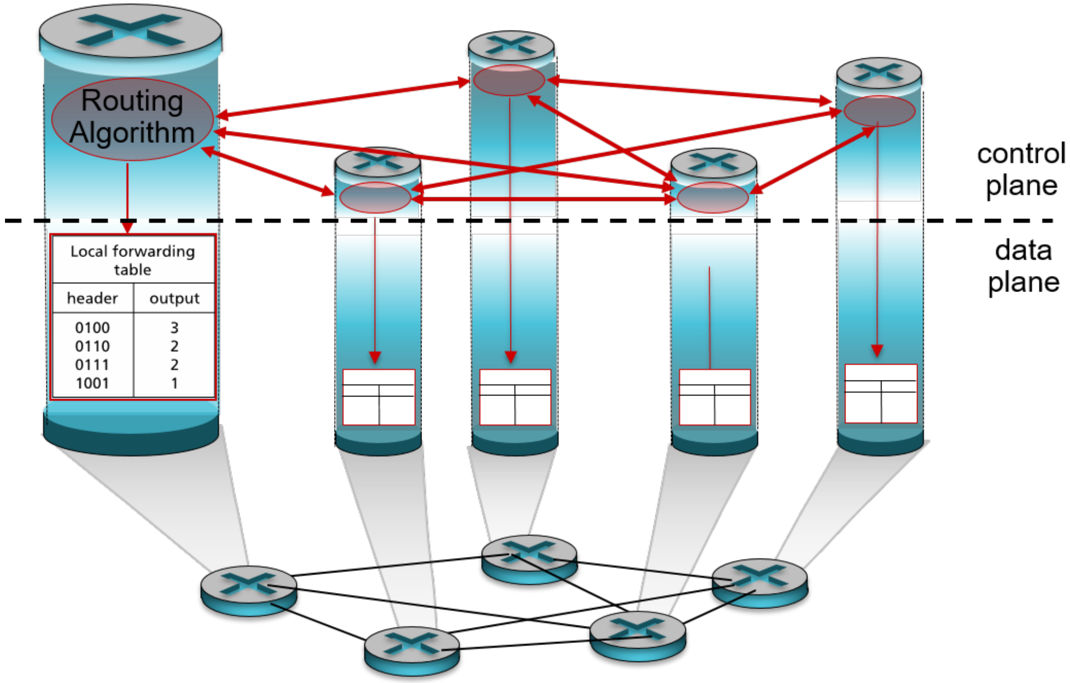
routing: determine route taken by packets from source to destination — **control plane**

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

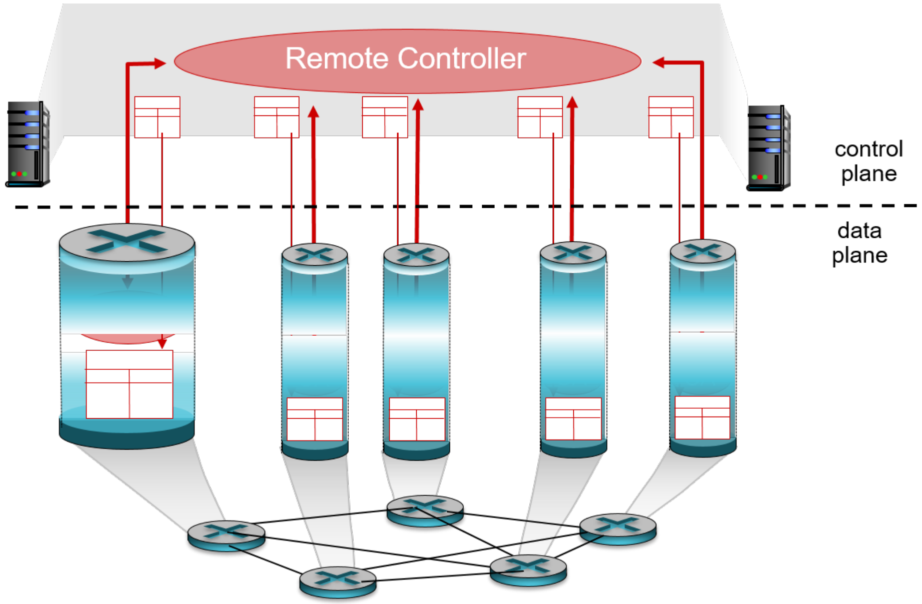
Per-Router Control Plane

Individual routing algorithm components **in each and every router** interact with each other in control plane to compute forwarding tables



Logically Centralized Control Plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables

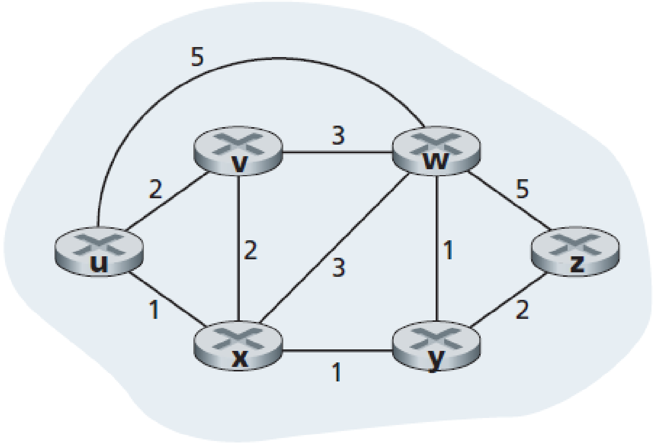


Routing Protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets will traverse in going from given initial source host to given final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!

Graph Abstraction of the Network



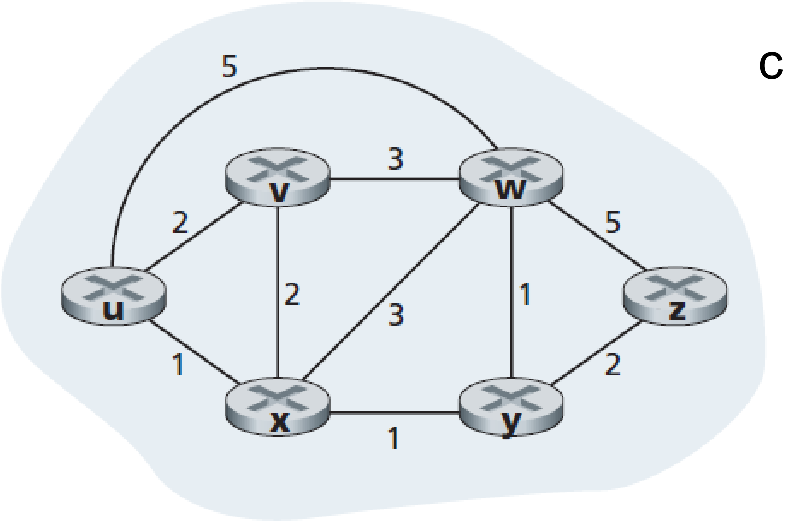
graph: G = (N, E)

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) }

aside: graph abstraction is useful in other network contexts, e.g., P2P, where **N** is set of peers and **E** is set of TCP connections

Graph Abstraction: Costs



$c(x,x') = \text{cost of link } (x,x')$ e.g., $c(w,z) = 5$

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

$$\text{cost of path } (x_1, x_2, x_3, \ldots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p)$$

key question*:* what is the least-cost path between u and z?

routing algorithm*:* algorithm that finds that least cost path

Routing Algorithm Classification

Q: 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**

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

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

A Link-State Routing Algorithm

Dijkstra's algorithm

- network topology, link costs: known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have the same information
- compute the least-cost paths from one node ("source") to all other nodes
  - gives **forwarding table** for that node
- iterative: after k iterations, know the least-cost paths to k destinations

Notation:

- **c(x,y):** link cost between node x and node y
- If **(x,y)** does not belong to E (that is, x and y are not direct neighbours), then we set **c(x,y)** = ∞
- **D(v):** current cost of the least-cost path from the source to destination **v** (as of this iteration of the routing algorithm)
- **p(v):** previous node (neighbour of **v**) along the current least-cost path from the source to **v**
- **N':** a subset of nodes whose least-cost path is definitively known

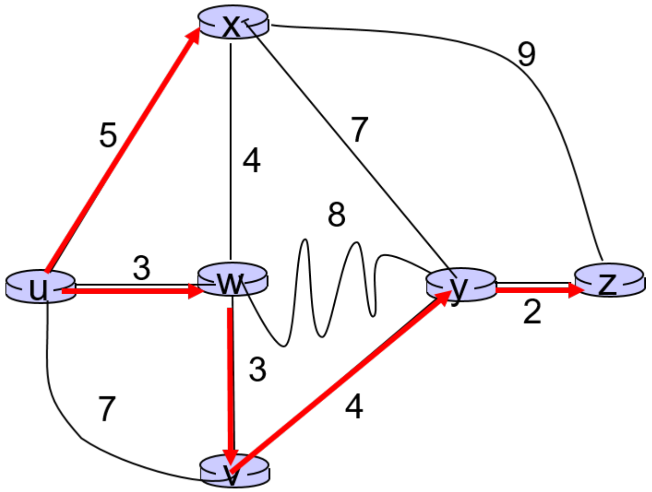
Dijsktra’s Algorithm

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

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'   (that is, N’=N)
```

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					

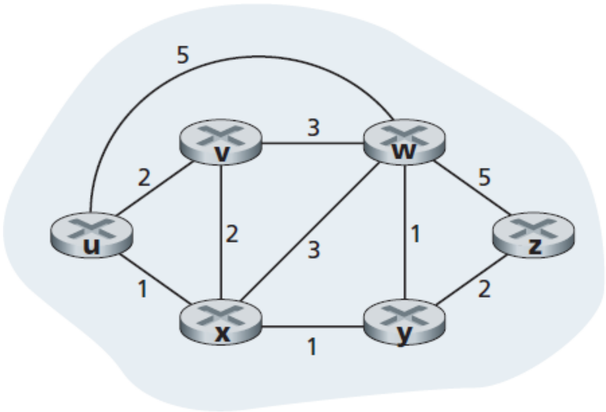


Notes:

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

Dijkstra's Algorithm: Another Example

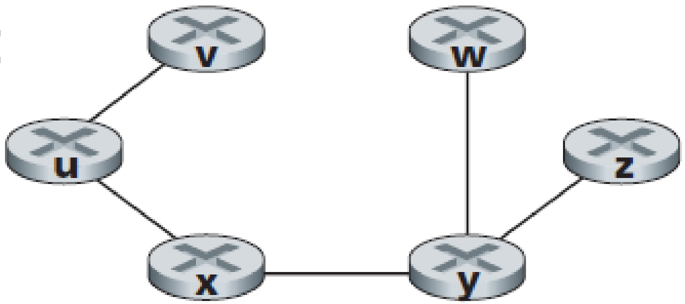
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Dijkstra’s Algorithm: Example (2 of 2)

- The resulting **least-cost paths** from u:



- The resulting **forwarding table** in u:

Destination	Link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Dijkstra's Algorithm: Discussion

Computational complexity: given *n* nodes (not counting the source)

- First iteration: need to check all *n* nodes to find the node *w* not in N' that has the minimum cost.
- Second iteration: check *(n-1)* nodes to determine the minimum cost
- And so on…
- Overall, $n(n+1)/2$ comparisons. Worst−case complexity: $O(n^2)$
- More efficient implementation (using heap data structure) to reduce complexity to $O(n\log(n))$

Distance Vector Algorithm (1 of 6)

Bellman-Ford equation (dynamic programming)

let

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

then

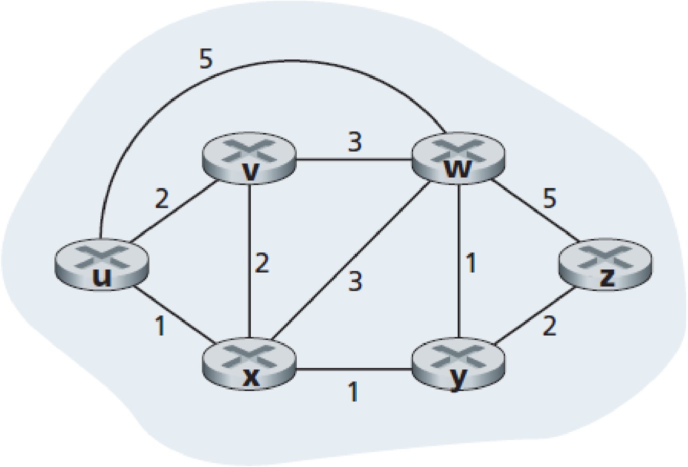
$$d_x(y) = \min_v \{\underbrace{c(x,v)}_{\text{cost to neighbor } v} + \underbrace{d_v(y)}_{\text{cost from neighbor } v \text{ to destination } y}\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford Example



clearly, $d_v(z) = 5,\ d_x(z) = 3,\ d_w(z) = 3$

B-F equation says:

$$
\begin{aligned}
d_u(z) = \min \{ & c(u,v) + d_v(z), \\
& c(u,x) + d_x(z), \\
& c(u,w) + d_w(z) \} \\
= \min \{ & 2 + 5, \\
& 1 + 3, \\
& 5 + 3 \} = 4
\end{aligned}
$$

Node achieving minimum is the next hop in the shortest path, used in the forwarding table

Distance Vector Algorithm (2 of 6)

Initialisation step:

- Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from x to y, for all nodes y in N.
- Let $\mathbf{D}_x = [D_x(y)\text{: } y \text{ in } N]$ be node x's distance vector of cost estimates from x to all other nodes y in N.
- Each node x maintains the following routing information:
  - The cost c(x,v) from x to each directly attached neighbor v
  - Node x's distance vector $\mathbf{D}_x = [D_x(y)\text{: } y \text{ in } N]$
  - The distance vectors of each of its neighbours v, that is, $\mathbf{D}_v = [D_v(y)\text{: } y \text{ in } N]$

Distance Vector Algorithm (3 of 6)

Update step:

- From time-to-time, each node sends a copy of its own distance vector to its neighbours
- When a node x receives a new distance vector from its neighbor w, it saves w's distance vector and updates its own distance vector using B-F equation: $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- If node x's distance vector has changed as a result of this update step, node x sends its updated distance vector to each of its neighbours, which in turn update their own distance vectors.
- Miraculously, each cost <u>**estimate**</u> $D_x(y)$ ***converges*** to $d_x(y)$, the <u>**actual**</u> cost of the least-cost path from node x to node y!

Distance Vector Algorithm (4 of 6)

iterative, asynchronous*:* each local iteration caused by:

- local link cost change
- Distance vector update message from neighbour

distributed:

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

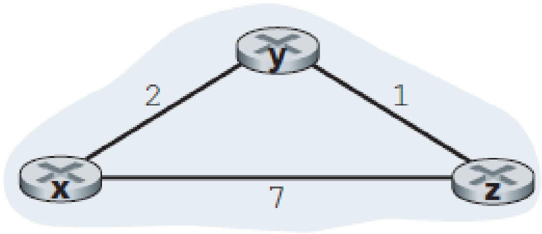
each node:

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

recompute estimates

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

Distance Vector Algorithm (5 of 6)



$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

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

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

node y table

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

node z table

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

time

Distance Vector Algorithm (6 of 6)

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

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

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

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

node y table

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

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

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

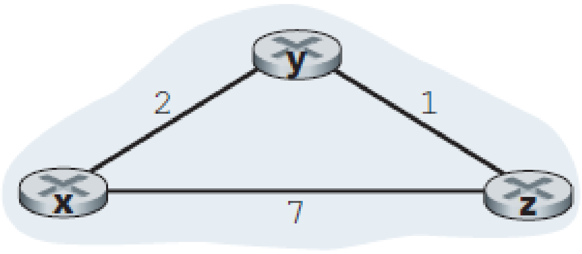
node z table

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

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

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

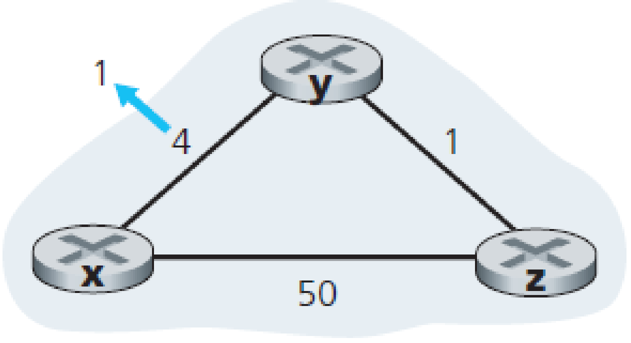
time



Distance Vector: Link Cost Changes (1 of 2)

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.

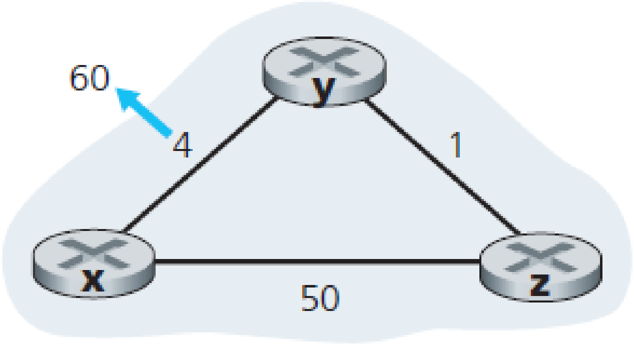
t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does **not** send a message to *z*.

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance Vector: Link Cost Changes (2 of 2)

link cost changes:

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



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? No – for loops involving three or more nodes.

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