Link State routing protocol

CE 352, Computer Networks

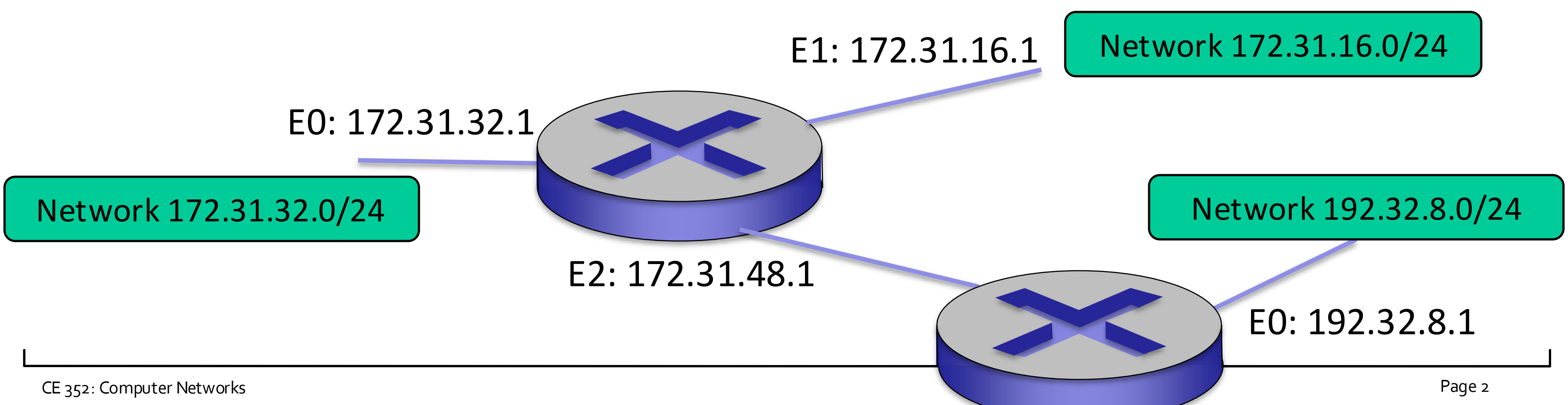
Salem Al-Agtash

Lecture 15

Slides are adapted from Computer Networking: A Top Down Approach, 7th Edition © J.F Kurose and K.W. Ross

Recall (Network-layer functions)

- *Data plane - forwarding:* move packets from router's input to appropriate router output
- *Control plane - routing:* determine route taken by packets from source to destination:
  - per-router control (traditional)
    - routers exchange information between each other
    - Individual router creates a forwarding table
  - logically centralized control (software defined networking - SDN)



Network layer control plane

Control plane

- Traditional routing algorithms: Link state and distance vector
- SDN controllers

Implementation in the Internet:

- OSPF: Open Shortest Path First
- BGP: Border Gateway Protocol
- ICMP: The Internet Control Message Protocol
- SNMP: Simple Network Management Protocol
- SDN OpenFlow and ONOS controllers

Internet routing protocols are responsible for constructing and updating the forwarding tables at routers

Inside router

Router processor

- Loopback interface: IP address of the CPU on the router for internal testing and diagnosis
- Administrator interface: Command line interface for configuration and monitory
- Implementing routing protocols
- Setting up forwarding tables

Interfaces

- Ethernet (Twisted Pair): 10BaseT → 10Mbps
- Fast Ethernet (Twisted Pair): 100BaseT → 100 Mbps
- Gigabit Ethernet (Twisted Pair): 1000BaseT → 1Gbps
- Gigabit Ethernet (Fiber SFP): 1000Base-SX (Multimode) – LX(Single mode)
- Serial: T1, T2
- FDDI: Fiber Distributed Data Interface – 100 Mbps
- Token Ring: 4Mbps – 16 Mbps

1000Base-T RJ45

40GBase-SR4 SFP

Forwarding tables

- Computed forwarding tables: Map IP prefix (network ID) to outgoing links
- Statistically configured: ”map 172.31.48.0/24 to FastEthernet0/1”

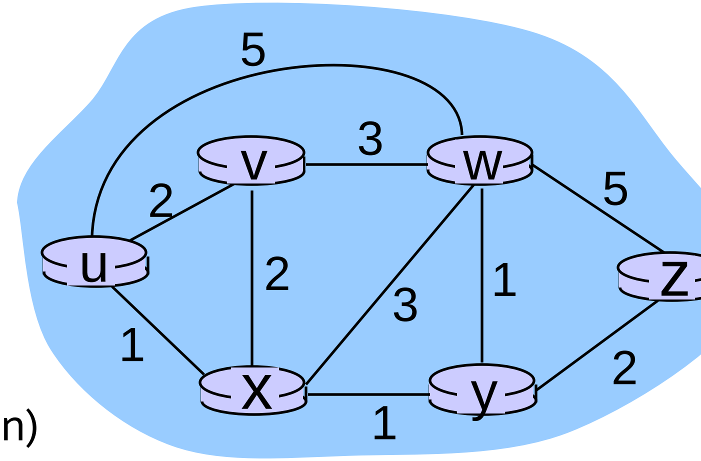
Routing protocols

Routing protocols implement the core function of a network

- determine "good" paths (equivalently, routes), from sending hosts to receiving host, through a network of routers

Network modeled as a graph

- graph: G = (N,E)
- N = set of nodes (routers) = { u, v, w, x, y, z }
- E = set of edges (links) ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }
- Edges have costs (distance, bandwidth, congestion)

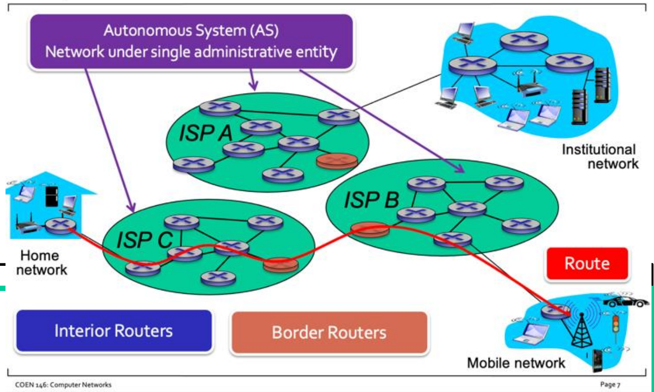


Path: sequence of routers packets will traverse in going from given initial source host to given final destination host

"good" path: least "cost", "fastest", "least congested" routing

Internet routing

Important context



Routers

- Need to know which router to use to reach a destination prefix (network ID)
- Need to know with outgoing interface to use to reach that router

Intra-domain

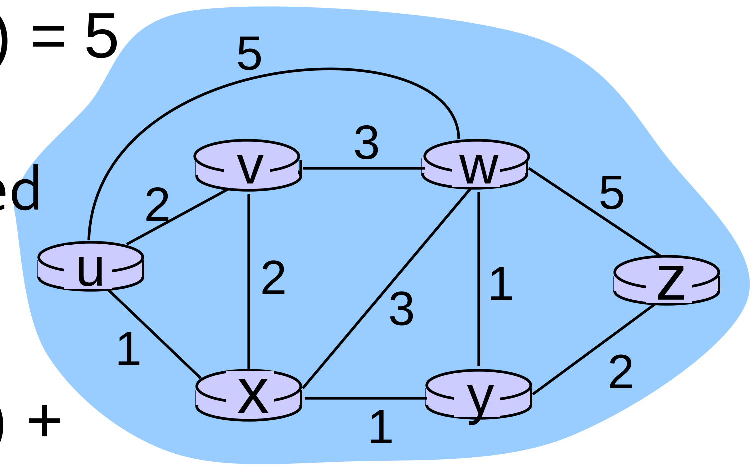
- Each "Autonomous System - AS" network (e.g. ISP) runs Intra-domain routing protocol to setup routes within its domain
  - Link State: e.g. Open Shortest Path First (OSPF)
  - Distance Vector: e.g. Routing Information Protocol (RIP)

Inter-domain

- AS networks run inter-domain routing protocols to setup routes between domains
  - Path Vector: Border Gateway Protocol (BGP)

Graph abstraction: costs

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



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

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

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:

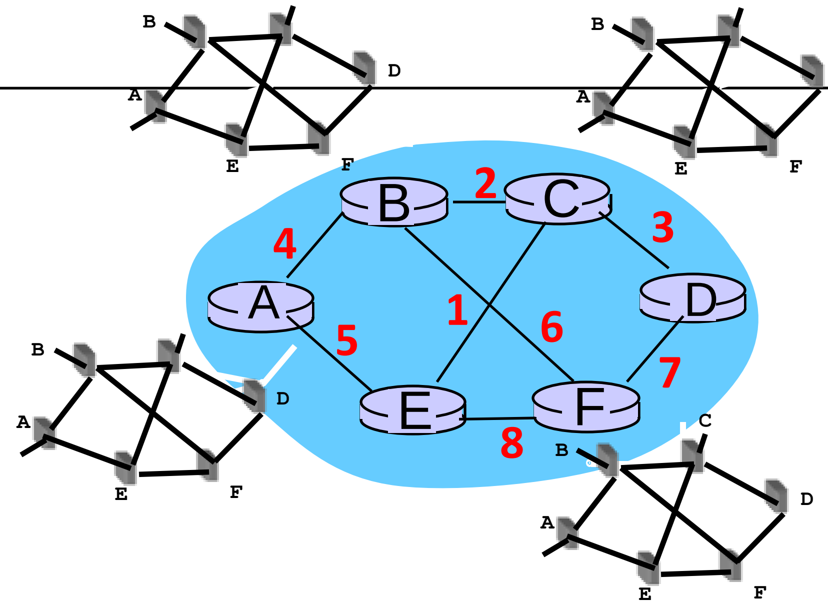
- routes change slowly over time

dynamic:

- routes change quickly periodic, update in response to cost changes

Link-state routing

- Each node maintains local link state
  - List of directly attached links and costs (Echo packet), sequence #, and Age
  - Construct LS packet
- Each node floods its local link state
  - When a router receives a new LS, it forwards the message to all its neighbors (excluding sending router)
- Each node learns the network topology
- Each node uses Dijkstra to compute the shortest paths between nodes



Link State Packets

A	
Seq.	
Age	
B	4
E	5

B	
Seq.	
Age	
A	4
C	2
F	6

C	
Seq.	
Age	
B	2
D	3
E	1

D	
Seq.	
Age	
C	3
F	7

E	
Seq.	
Age	
A	5
C	1
F	8

F	
Seq.	
Age	
B	6
D	7
E	8

- Sequence number (32 bits) is used to label LS packets
- Age (Time-to-Live) is used and when becomes 0, packet is discarded

Dijkstra algorithm

Shortest path algorithm

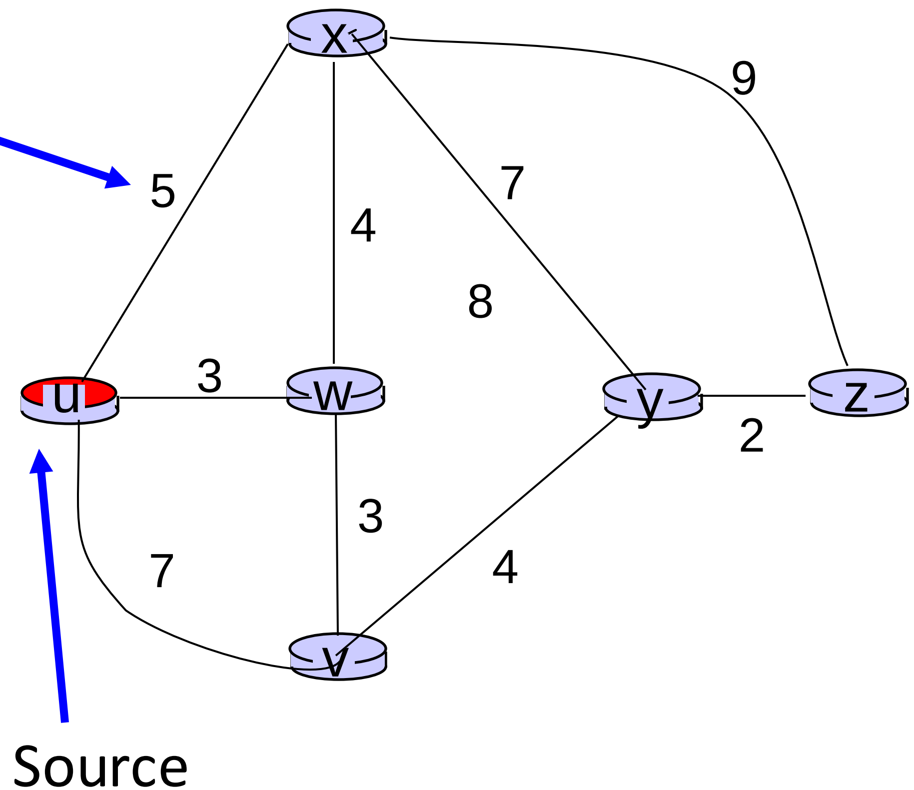
- Input
  - Net topology, link costs known to all nodes
    - "link state broadcast" → all nodes have same info

- Output
  - Least cost paths from one node ('source") to all other nodes
    - gives *forwarding table* for that node

- Iterative:
  - after k iterations, a node knows least cost path to its k closest neighbor

Notation

- c(x,y): link cost from node x to y; = ∞ 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
- N': set of nodes whose least cost path definitively known



Dijkstra algorithm

- c(x,y): link cost from node x to y; = ∞ 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
- N': set of nodes whose least cost path definitively known

Initialization:

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

Loop

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

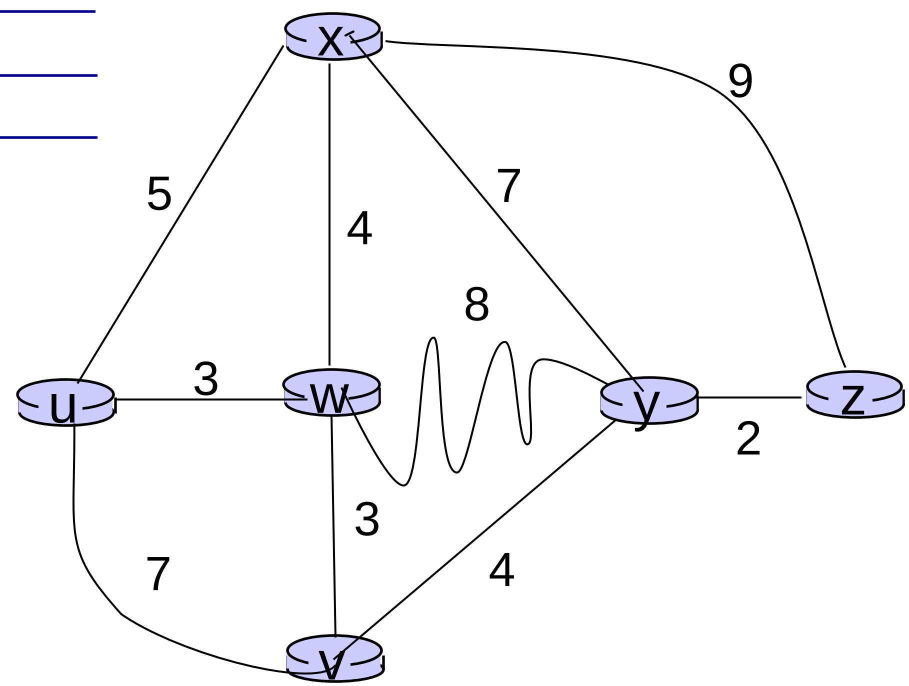
until all nodes in N'

Example 1

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						
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) = ∞

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



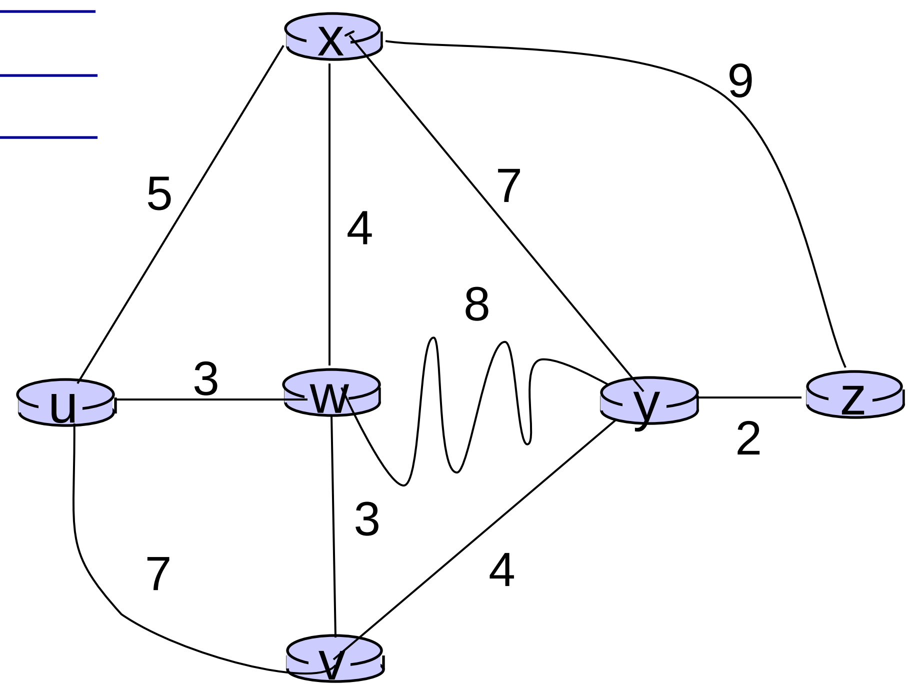
Example 1

w

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					
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) = ∞

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



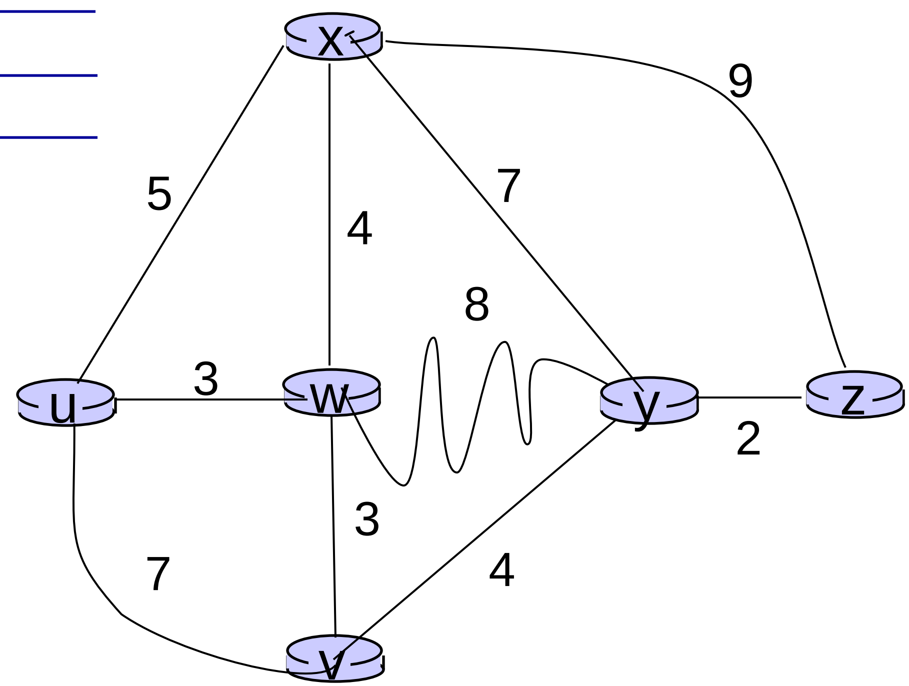
Example 1

w

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						
3						
4						
5						

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

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



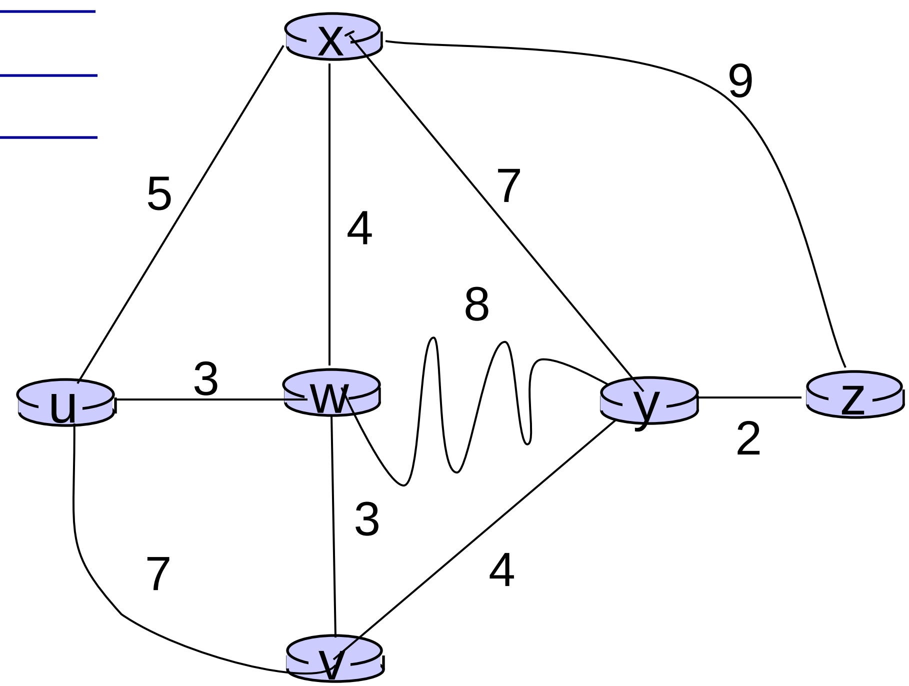
Example 1

w x

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					
3						
4						
5						

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

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



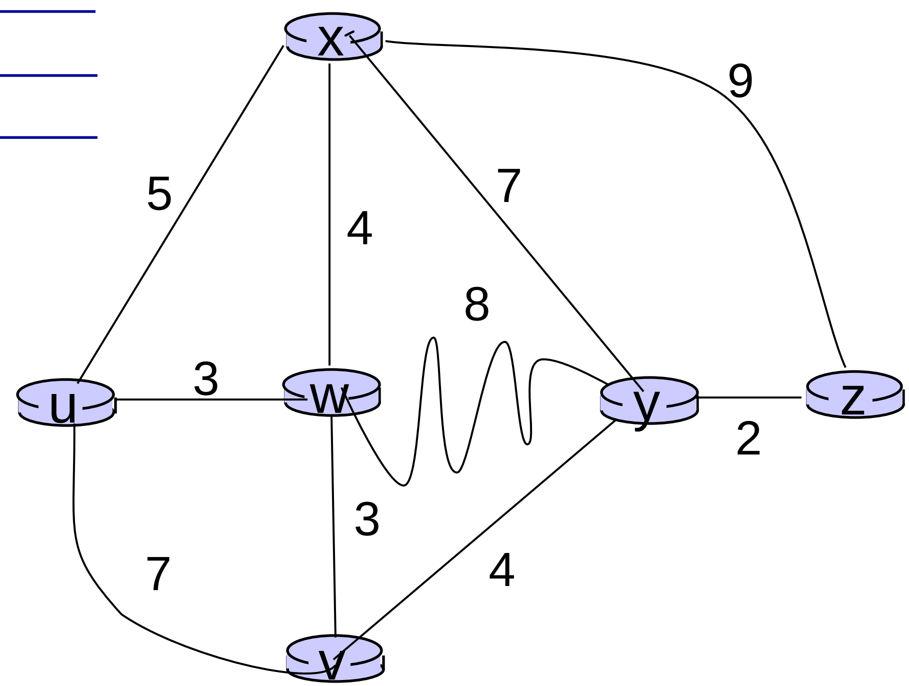
Example 1

v w x

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					
4						
5						

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

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



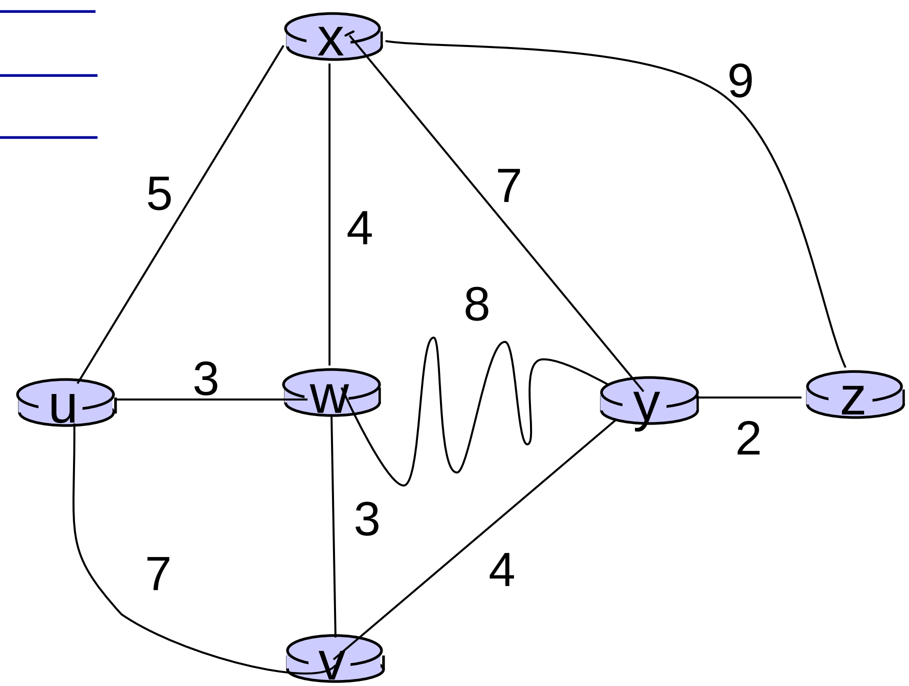
Example 1

v w x

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						
5						

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

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



Example 1

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					
5						

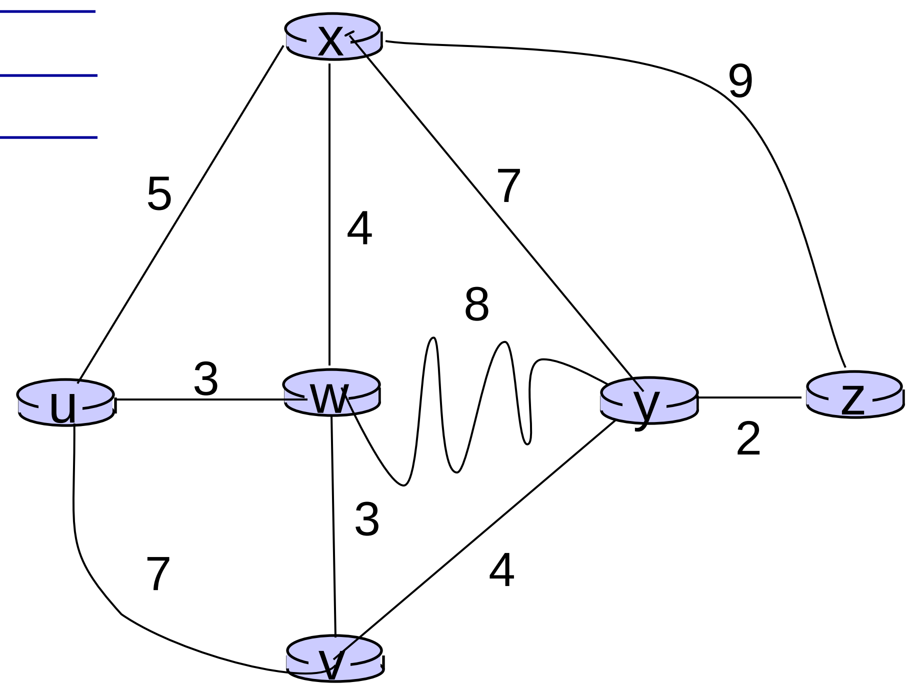
Initialization:

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

Loop

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

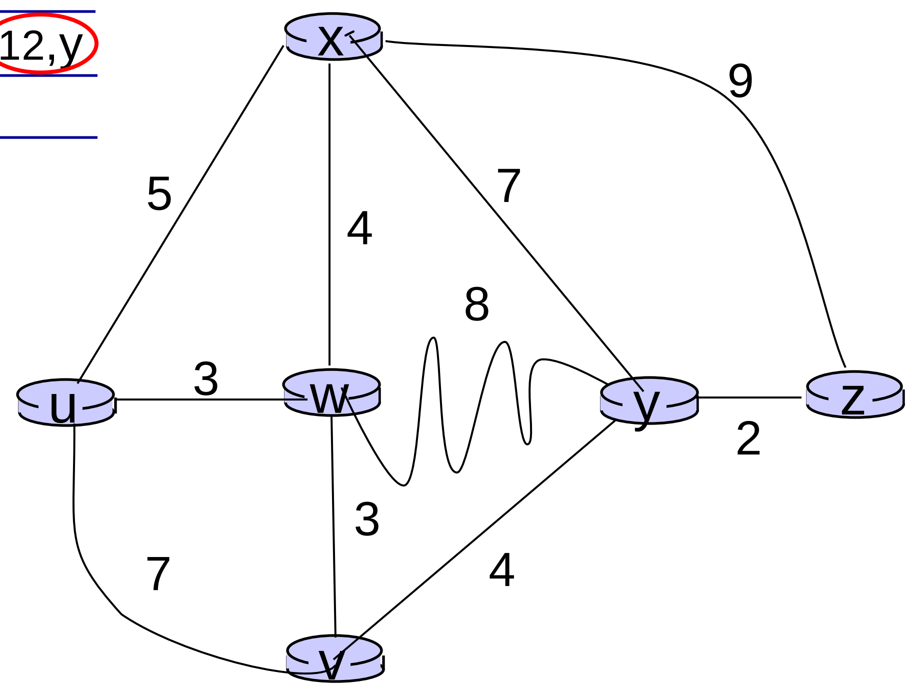
until all nodes in N'



Example 1

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						

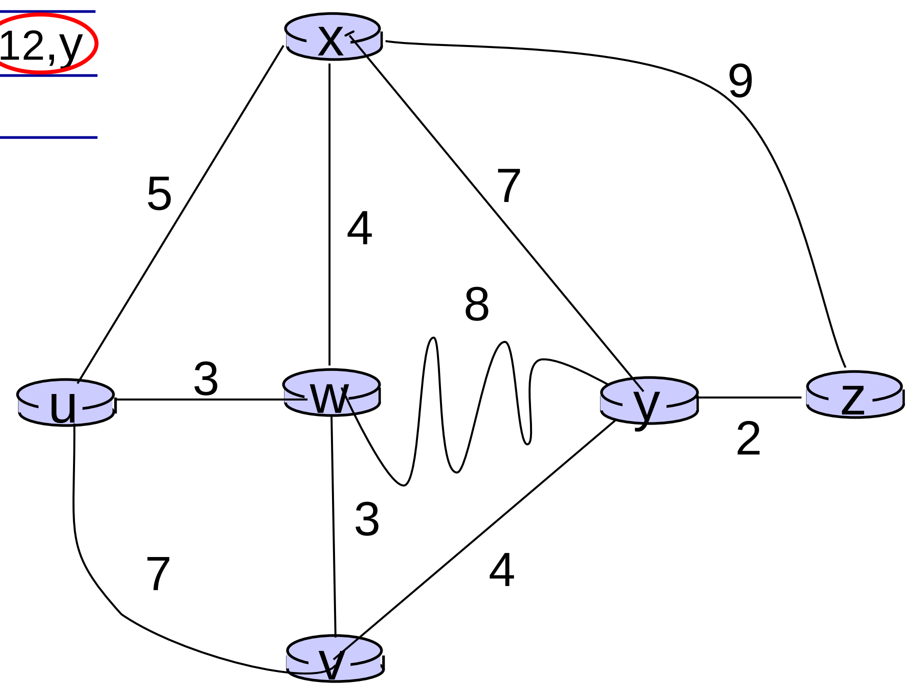
```
Initialization:
    N' = {u}
    for all nodes v
      if v adjacent to u
        then D(v) = c(u,v)
      else D(v) = ∞
Loop
    find node not in N' such that D(node) is a minimum
    add node to N'
    update D(v) for all v adjacent to node and not in N' :
      D(v) = min( D(v), D(node) + c(node,v) )
    /* new cost to v is either old cost to v or known
    shortest path cost to node plus cost from node to v */
until all nodes in N'
```



Example 1

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					

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
      then D(v) = c(u,v)
    else D(v) = ∞
Loop
  find node not in N' such that D(node) is a minimum
  add node to N'
  update D(v) for all v adjacent to node and not in N' :
    D(v) = min( D(v), D(node) + c(node,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to node plus cost from node to v */
until all nodes in N'
```

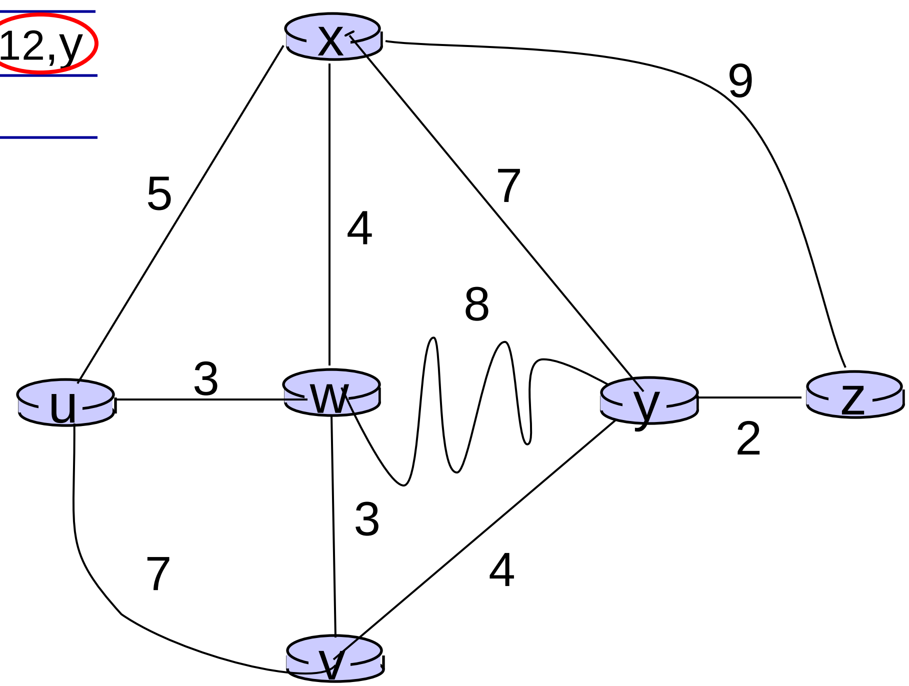


Example 1

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					

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

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



notes:

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

Example 1

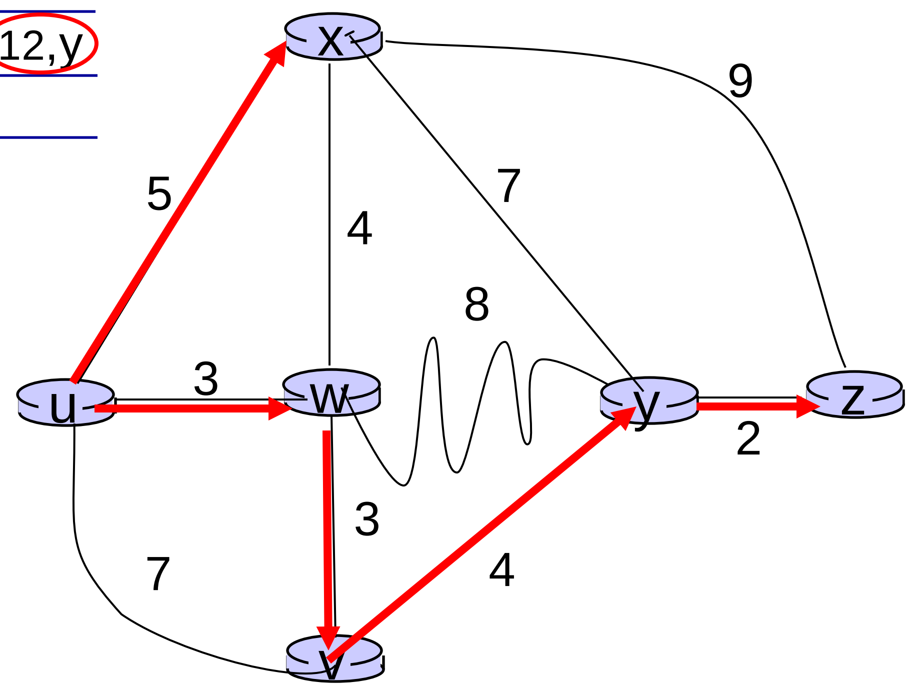
Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

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

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

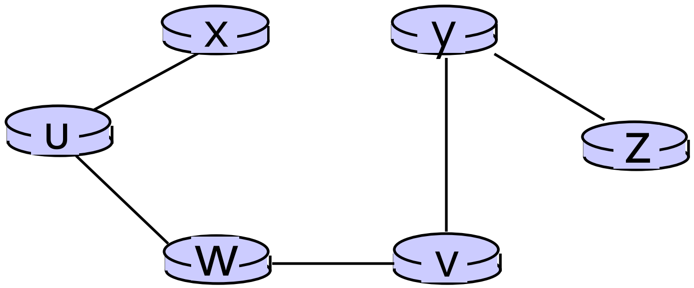
notes:

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



Forwarding table @ node u

Running Dijkstra at node U gives the following shortest path tree from u to all destinations

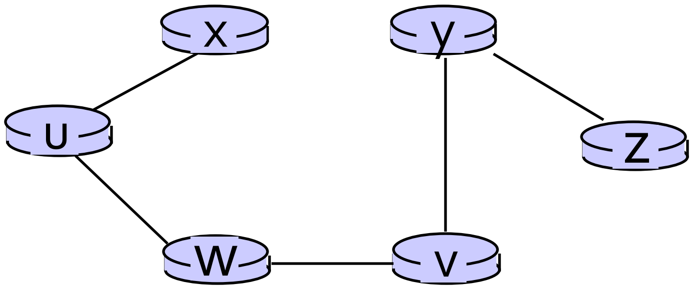


The forwarding table is constructed at u :

destination	link
v	?
x	?
y	?
w	?
z	?

Forwarding table @ node u

Running Dijkstra at node U gives the following shortest path tree from u to all destinations



The forwarding table is constructed at u :

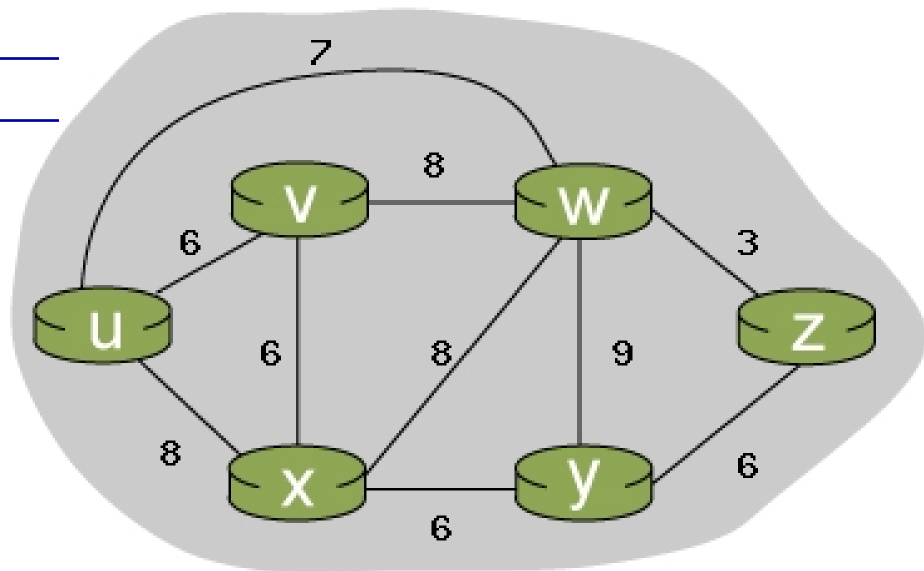
destination	link
v	(u,w)
x	(u,x)
y	(u,w)
w	(u,w)
z	(u,w)

Example 2

Step	N'	D(v), p(v)	D(x), p(x)	D(w), p(w)	D(y), p(y)	D(z), p(z)
0	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) = ∞

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

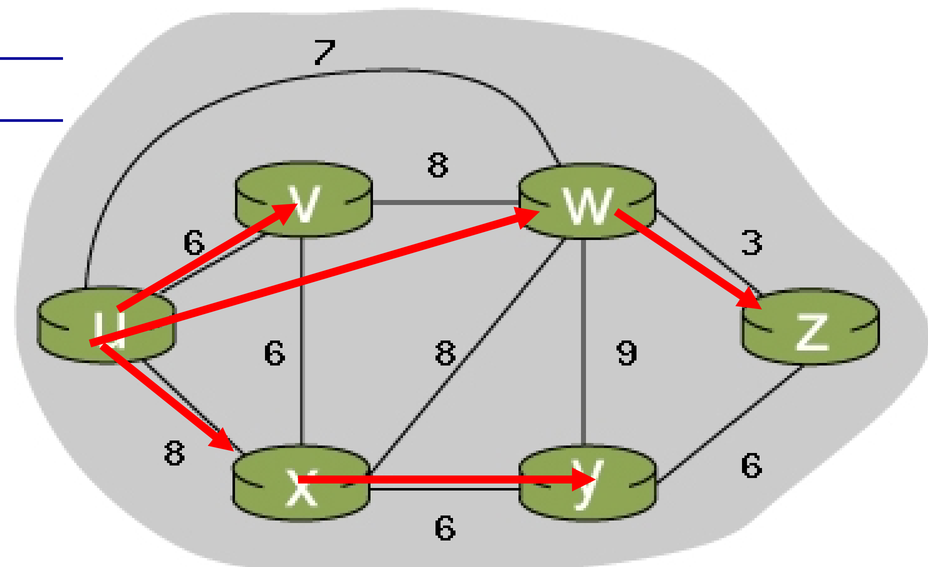


Example 2

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

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

Step	N'	D(v) p(v)	D(x) p(x)	D(w) p(w)	D(y) p(y)	D(z) p(z)
0	u	6,u	8,u	7,u	∞	∞
1	uv		8,u	7,u	∞	∞
2	uvw		8,u		16,w	10,w
3	uvwx				14,x	10,w
4	uvwxz				14,x	
5	uwxvyz					



notes:

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

Example 2

Initialization:

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

Loop

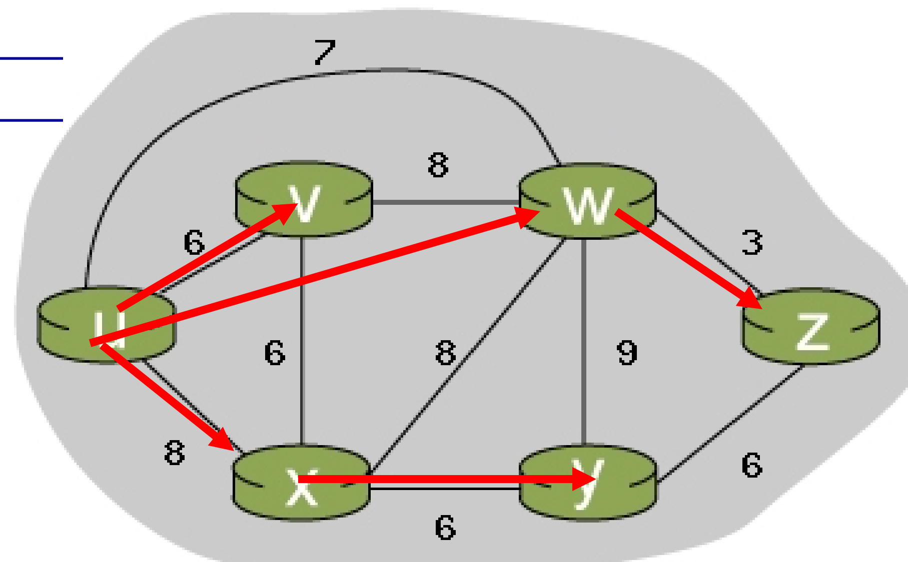
find **node** not in **N'** such that D(node) is a minimum
add **node** to **N'**
update D(v) for all **v** adjacent to **node** and not in **N'** :
  D(v) = min(D(v), D(node) + c(node,v))
/* new cost to **v** is either old cost to **v** or known
  shortest path cost to **node** plus cost from **node**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

until all nodes in N'

Step	N'	D(v) p(v)	D(x) p(x)	D(w) p(w)	D(y) p(y)	D(z) p(z)
0	u	6,u	8,u	7,u	∞	∞
1	uv		8,u	7,u	∞	∞
2	uvw		8,u		16,w	10,w
3	uvwx				14,x	10,w
4	uvwxz				14,x	
5	uwxvyz					

Forwarding table @ u

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

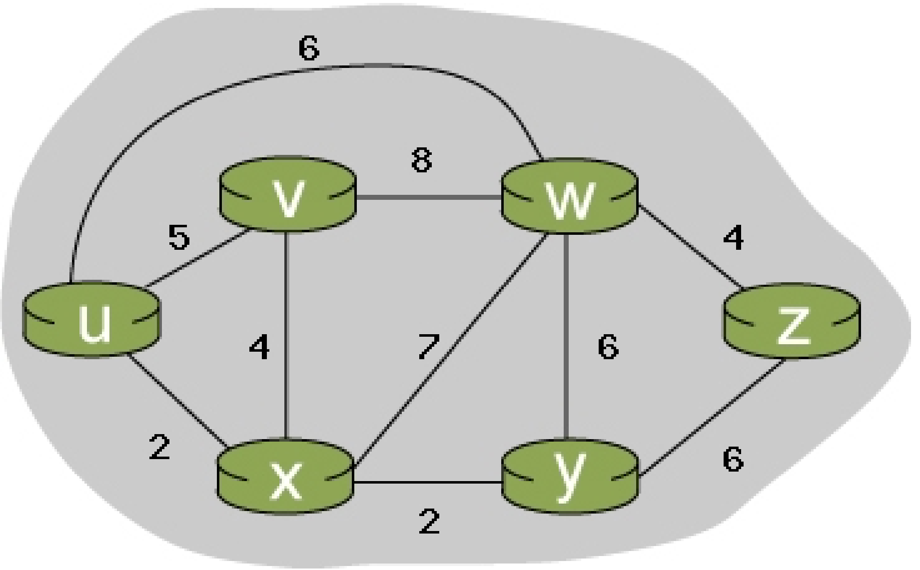


Result: u-to-v, u-to-w, u-to-x, x-to-y, w-to-z

Example 3

Step
0
1
2
3
4
5

Forwarding table @ u ?

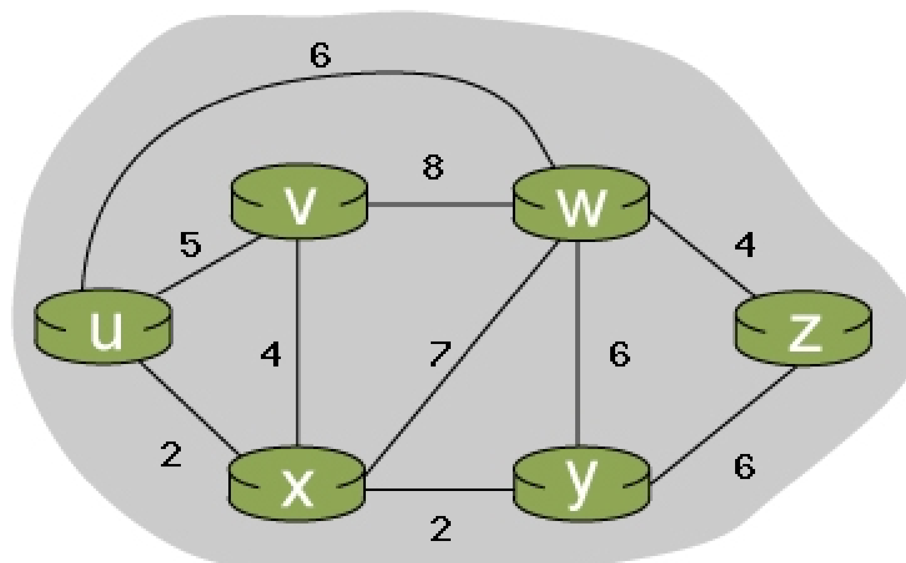


Initialization:
```
N' = {u}
for all nodes v
  if v adjacent to u
    then D(v) = c(u,v)
  else D(v) = ∞
```

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

Example 3

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



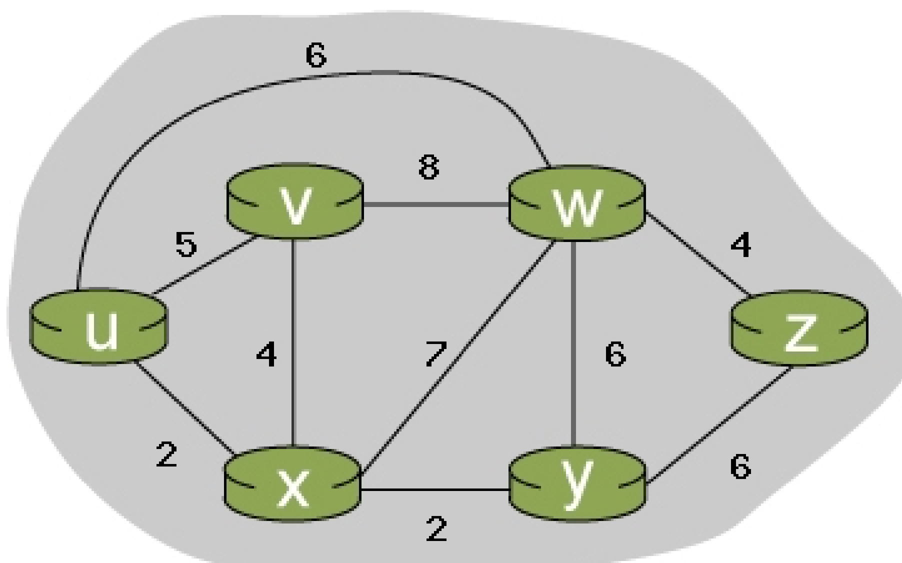
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
      then D(v) = c(u,v)
    else D(v) = ∞
Loop
  find node not in N' such that D(node) is a minimum
  add node to N'
  update D(v) for all v adjacent to node and not in N' :
     D(v) = min( D(v), D(node) + c(node,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to node plus cost from node to v */
until all nodes in N'
```

Example 3

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

Forwarding table @ u

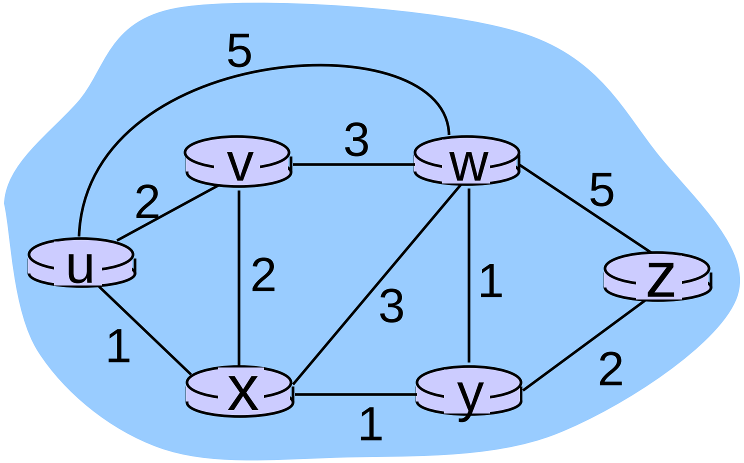
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,w)
z	(u,x)



Result: u-to-v, u-to-w, u-to-x, x-to-y, y-to-z

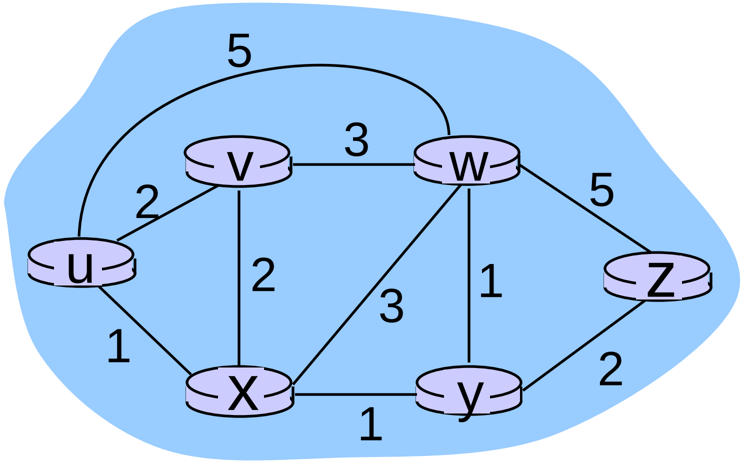
Example 4

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



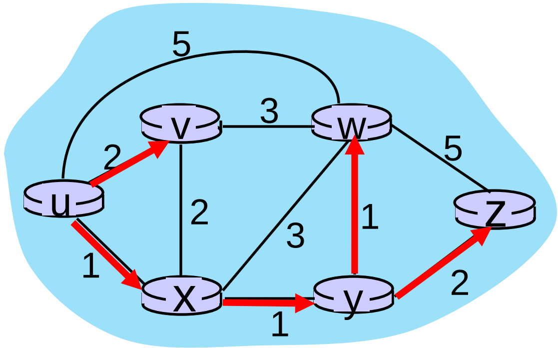
Example 4

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					



Example 4

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					



Initialization (step 0): For all a: if *a* adjacent to then

$D(a) = c_{u,a}$

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

add *a* to *N'*

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

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

Summary: Link-state routing

- Each router keeps track of its adjacent links (cost and operational status)
- Each router broadcasts the link state to give every router a complete view of the graph
  - Periodically every 30 minutes (e.g.), or topology change (Link/node failure or recovery)
  - Sequence number (32 bits) is used to label LS packets and keep flooding in check, when a new one arrives the older is discarded
  - Age (Time-to-Live) is used and when becomes 0, packet is discarded
- Each router runs Dijkstra's algorithm to compute the shortest path and construct the forwarding table
- Widely used protocol: OSPF

Dijkstra's algorithm complexity

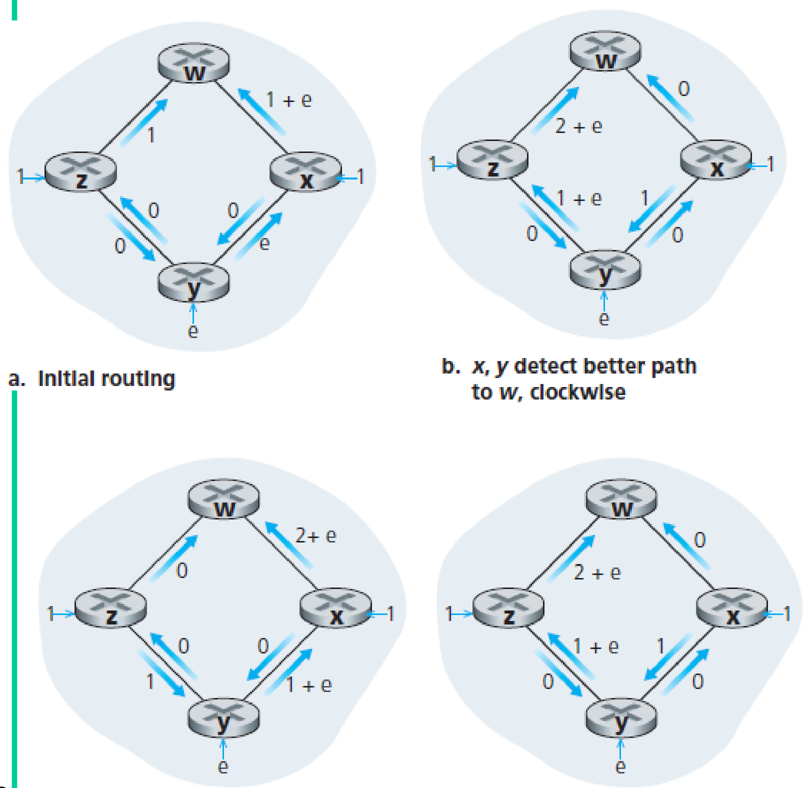
algorithm complexity for a network (graph) with n nodes and e edges:

- each iteration: need to check all nodes not in N'
- n(n+1)/2 comparisons: $O(n^2)$
- Flood link state messages: O(nxe)
- Entries in the forwarding table: O(n)
- Entries in the LS topology data bases: O(e)

Oscillation is possible

oscillations possible: Inconsistent link state database, leading to transient forwarding loops

- Link cost is equal to the load carried on the link
- z send 1 packet to w, y sends e packet to w, and x sends 1 packet to w
- Paths to w from x and y are favorable clockwise, and so generate new paths (figure b)
- When LS runs again detects 0 cost paths to w in the counterclockwise routes (figure c)
- Figure d
- Oscillation
  - Solution 1 – not to measure link based on traffic
  - Solution 2- ensure not all routers run LS same time



a. Initial routing

b. *x*, *y* detect better path to *w*, clockwise

c. *x*, *y*, *z* detect better path to *w*, counterclockwise

d. *x*, *y*, *z*, detect better path to *w*, clockwise

Summary

Today:

- Routing protocols
- Link state

Canvas discussion:

- Reflection
- Exit ticket

Next time:

- read 5.2.2 (DV) of K&R
- follow on Canvas! material and announcements

Any questions?