Lecture 18: Network Layer
Link State and Distance Vector Routing

COMP 332, Spring 2023

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - homework 7
     - written due Wed., programming due next Wed.
     - please look at Linux virtual machine instructions and do set-up
       - come see me if any issues: we can also spend time at end of next class

2. Addressing
   - usage in routing
   - how to get an IP address

3. Control plane aka where routing happens
   - overview
   - link state routing
   - distance vector routing

Takeaways from last time

Routing is done between blocks of addresses

- not easy to route on graph with 2^32 nodes

Data plane vs. control plane

- **data plane:** your traffic (data) is forwarded here
- **control plane:** routers exchange information to set-up routes to be used in forwarding

Subnet part and host part of address

- a.b.c.d/x, where x is # of bits in subnet part
- `11001000 00010111 00010000 00000000`

Addressing

USAGE IN ROUTING

Routers forward traffic to networks not hosts

Forwarding table

- does not contain row for every dest IP address
- instead computes routes between subnets (blocks of addresses)

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

What if address ranges don't divide up nicely?

Longest prefix matching

– use longest address prefix that matches destination address

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

Question

DA: 11001000 00010111 00010110 10100001 which interface?

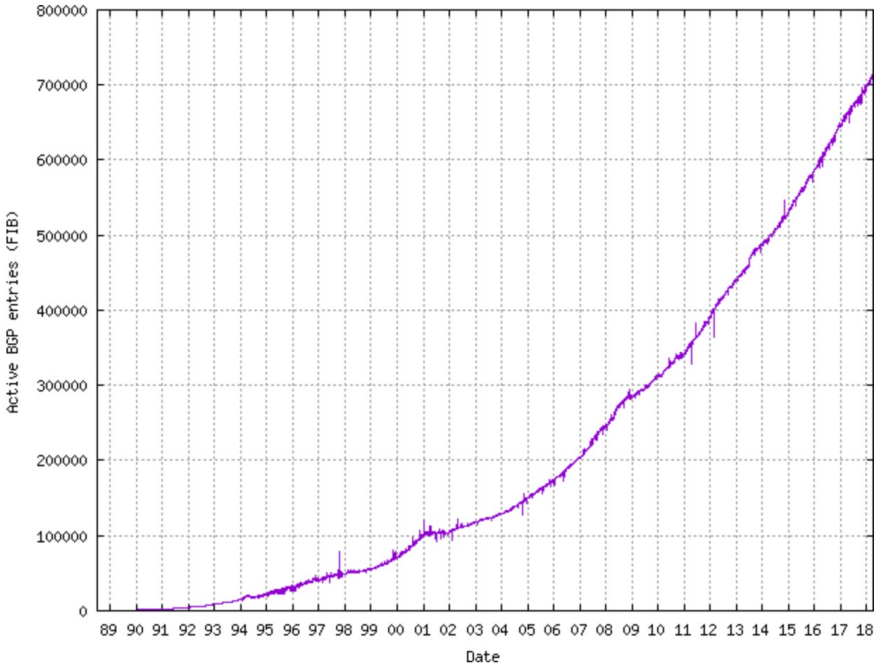
DA: 11001000 00010111 00011000 10101010 which interface?

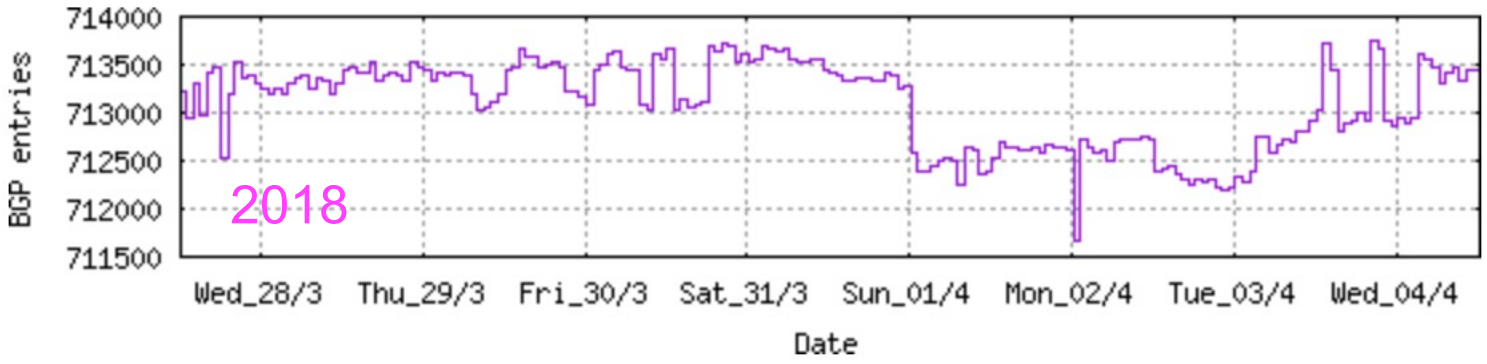
How big is a routing table for a core router?

From http://www.cidr-report.org/as2.0/

Table History

Date	Prefixes	CIDR Aggregated
28-03-18	713318	386580
29-03-18	713461	386983
30-03-18	713175	387365
31-03-18	713602	387141
01-04-18	713267	386331
02-04-18	712612	386192
03-04-18	712224	386045
04-04-18	712855	386936



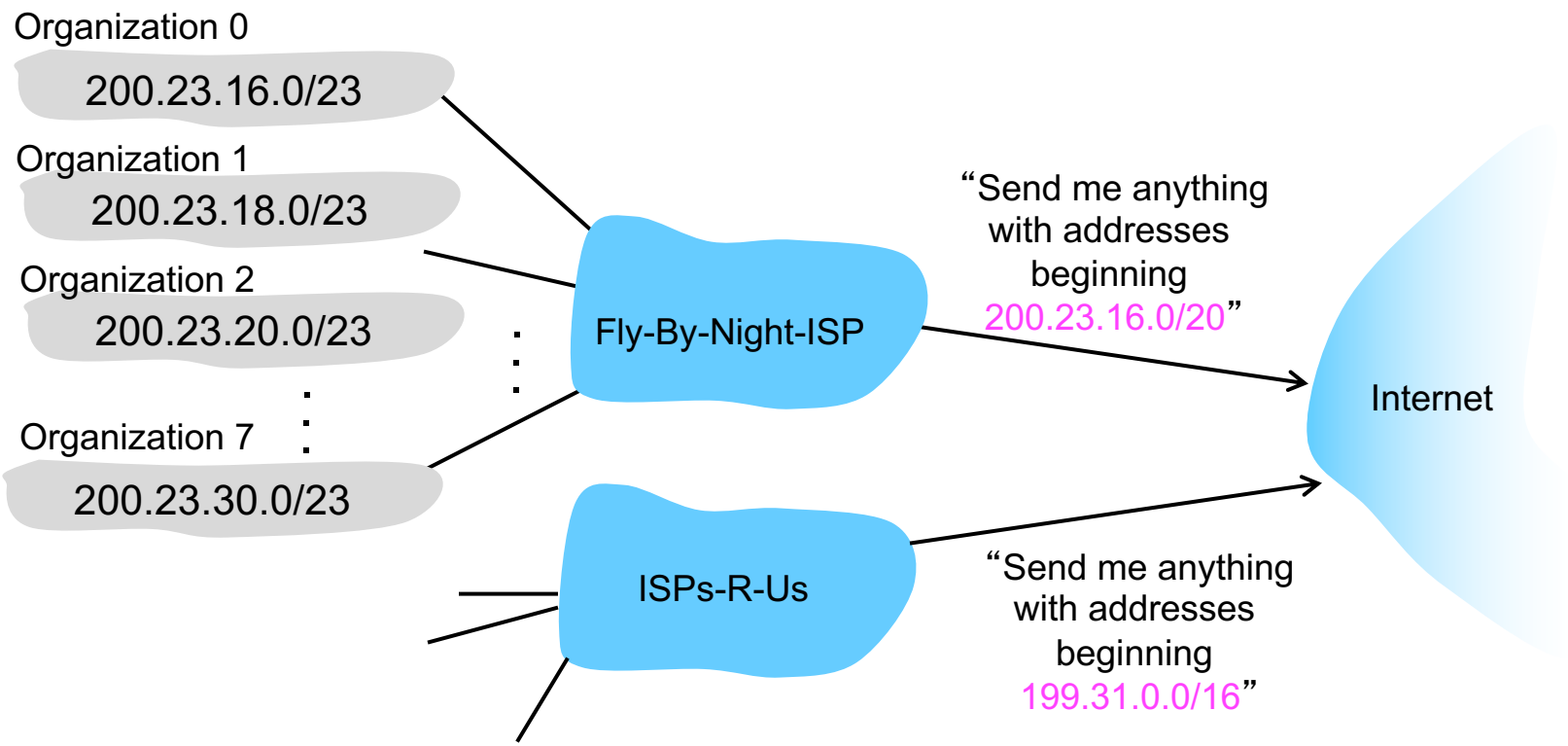


Q: If a core router processes 1million pkts+ per second, how fast does it need to be able to search table?

Hierarchical addressing

Route aggregation

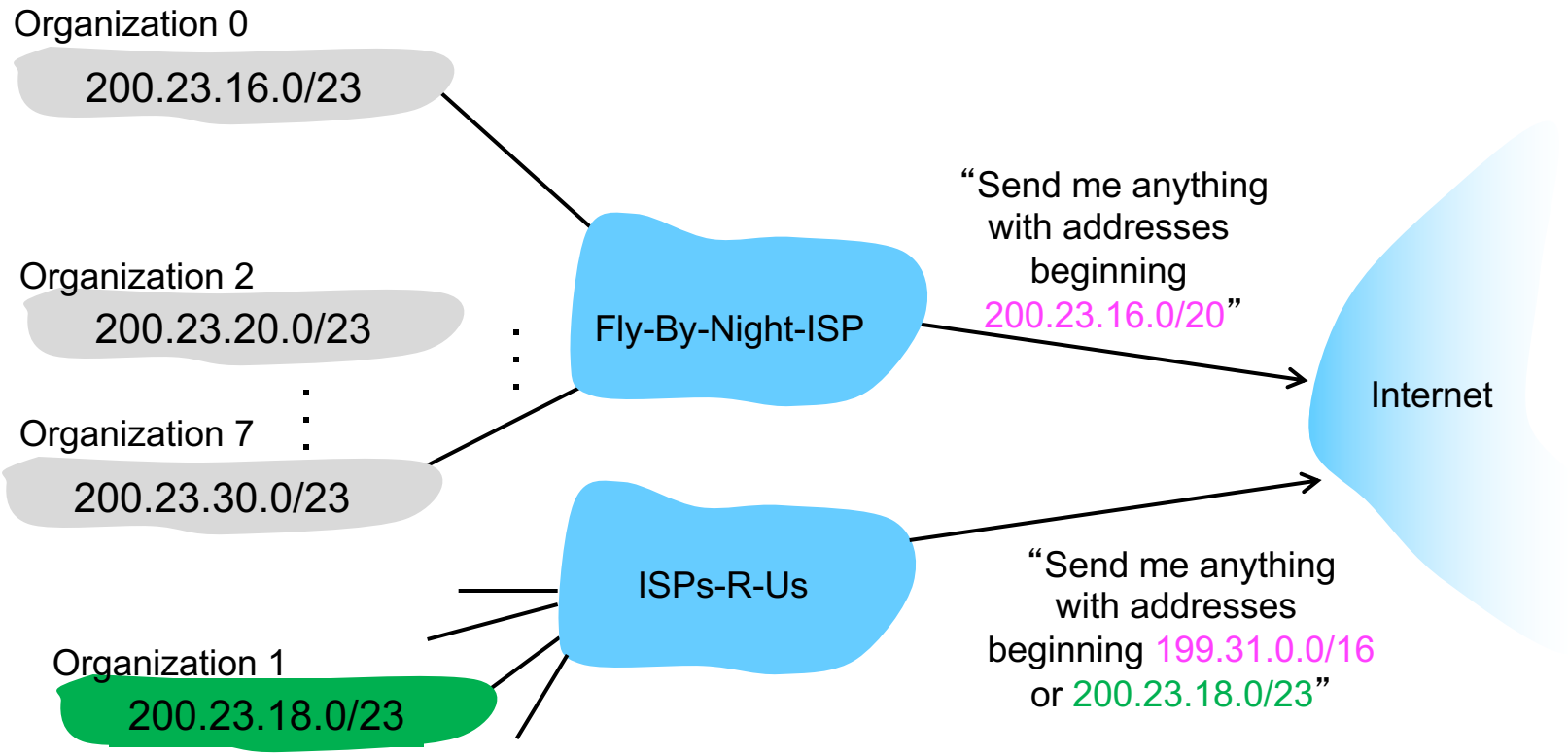
- combine multiple small prefixes into a single larger prefix
- allows efficient advertisement of routing information



Longest prefix matching

More specific routes

– ISPs-R-Us has a more specific route to Organization 1



Addressing

HOW TO GET AN IP ADDRESS?

How does ISP get block of addresses?

ICANN

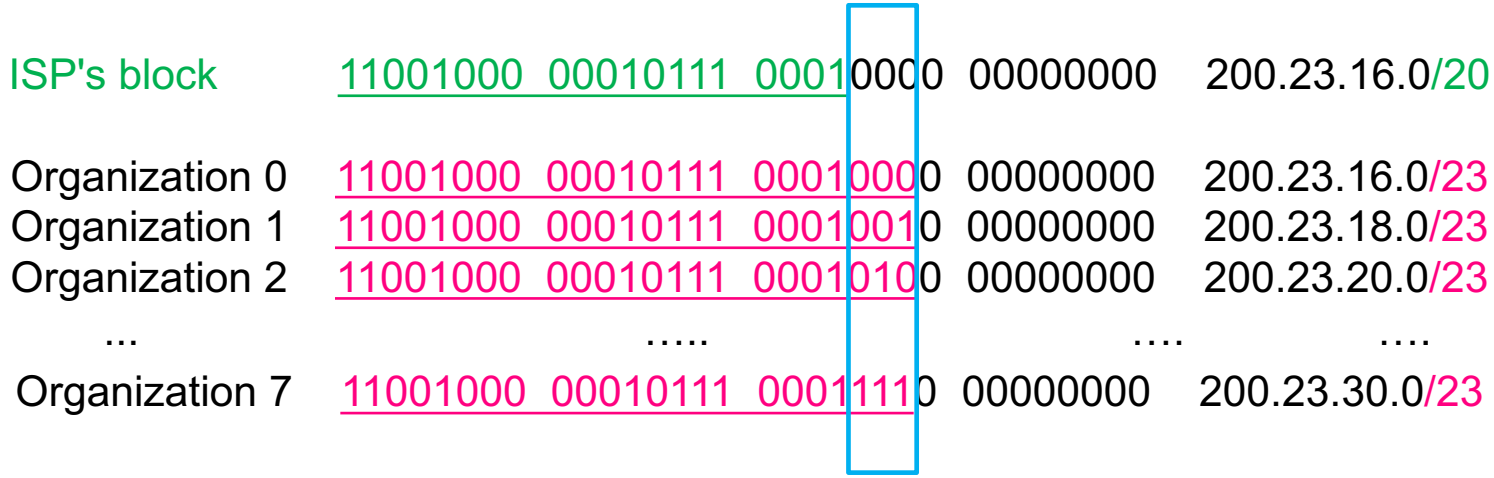
- Internet Corporation for Assigned Names and Numbers
- http://www.icann.org/

ICANN functions

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes
- …

How does network get net part of IP address?

Allocated portion of its provider ISP's address space



How does host get an IP address?

Option 1

- hard-coded by system admin in a file on your host

Option 2:

- dynamically get address from a server
  - DHCP: Dynamic Host Configuration Protocol

We're running out of IPv4 addresses

Why?

- inefficient use of address space
  - from pre-CIDR use of address classes (A: /8, B: /16, C: /24)
- too many networks (and devices)
  - Internet comprises 100,000+ networks
  - routing tables and route propagation protocols do not scale

Q: how many IPv4 addresses are there?

- 2^{32}

Solutions

- IPv6 addresses
- DHCP: Dynamic Host Configuration Protocol
- NAT: Network Address Translation

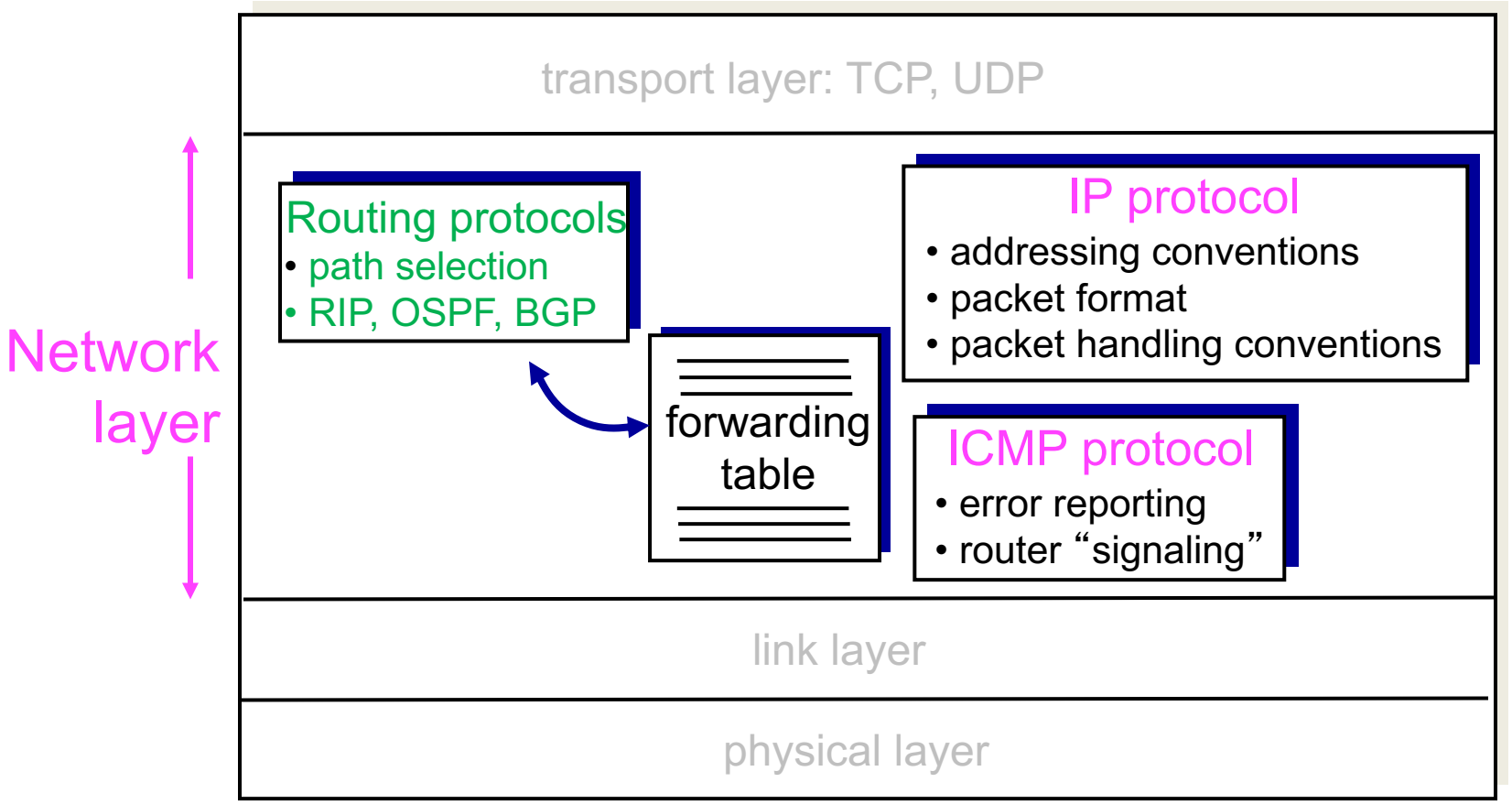
Control Plane

OVERVIEW

Internet's network layer

Network layer functions on hosts and routers

– control plane vs. data plane



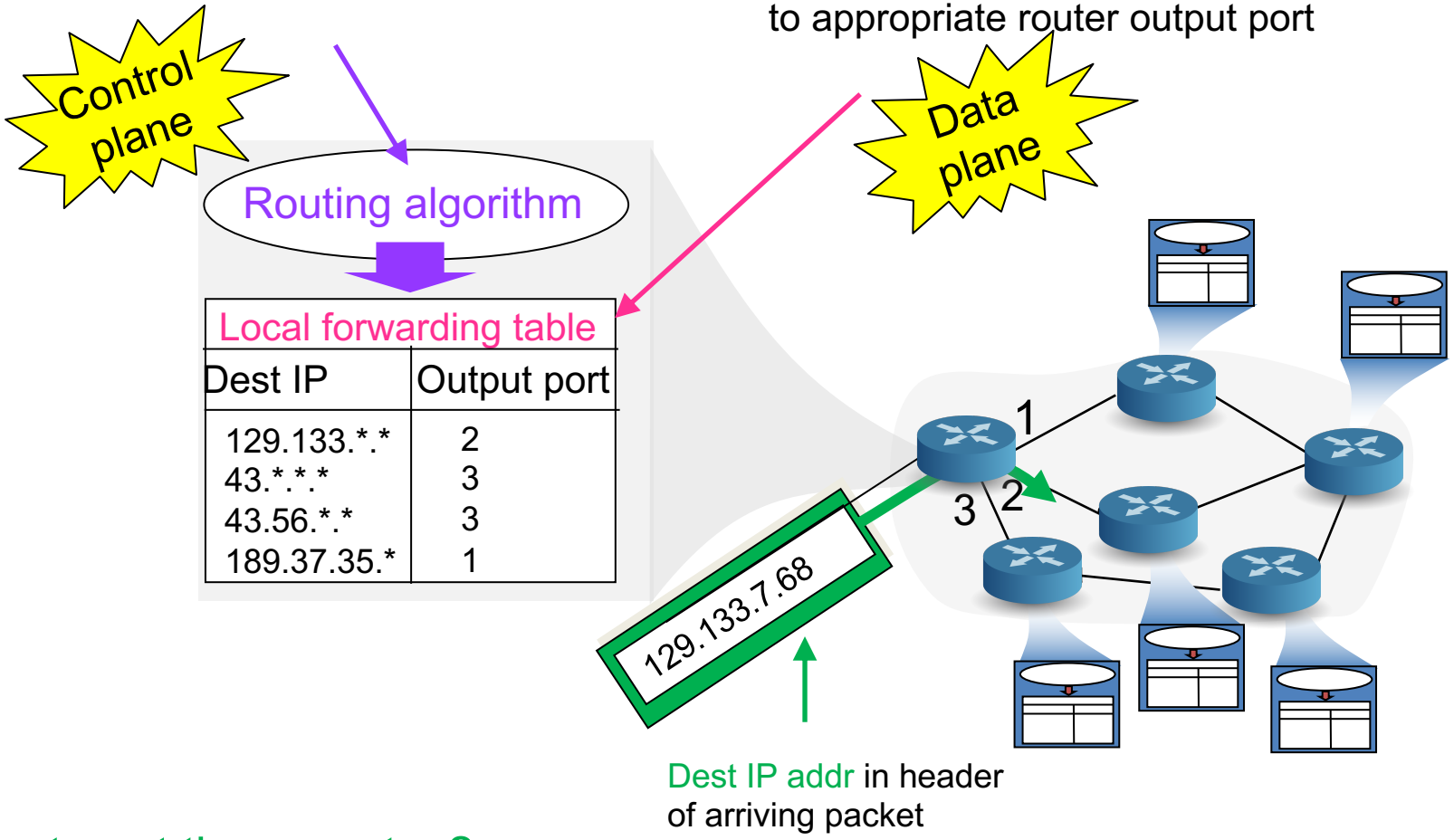
Control vs. data plane functions

Routing (slower time scale)

- determine route taken by packets from source to destination

Forwarding (faster time scale)

- move packets from router's input port to appropriate router output port



How to get these routes?

Routing protocols

Goal

- determine "good" path from sending hosts to receiving host, through network of routers

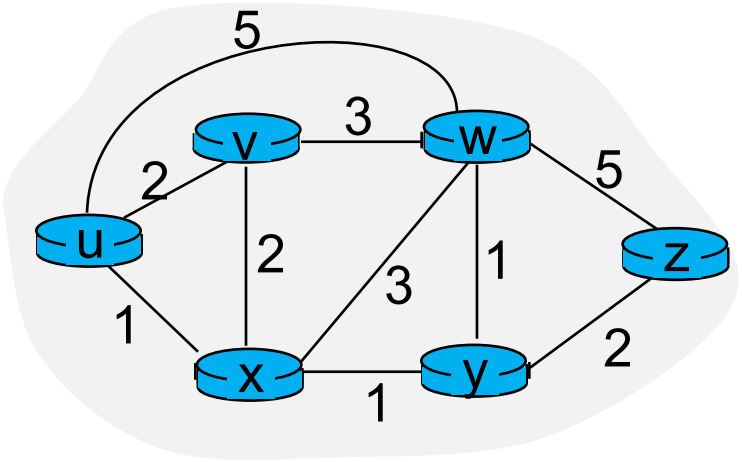
Path

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

"Good"

- least "cost", "fastest", "least congested", …
- correctness constraints
  - no loops
  - no dead-ends

Abstract network as a graph

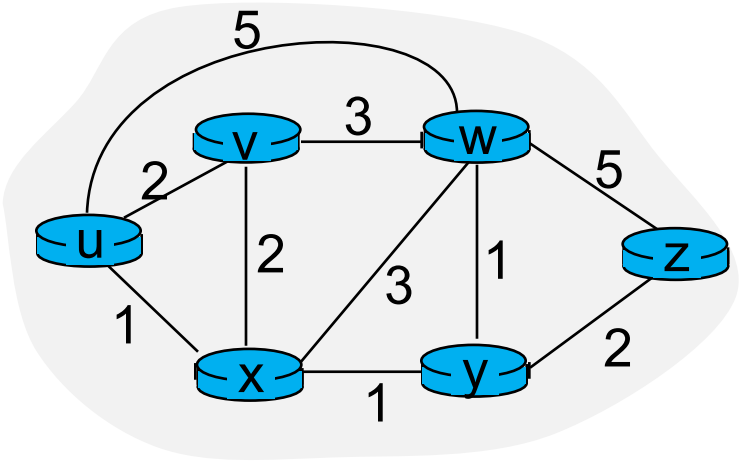


Graph: G = (N,E)

Q: What are the routers? I.e., nodes?

Q: What are the links? I.e., edges?

Abstract network as a graph



Graph: G = (N,E)

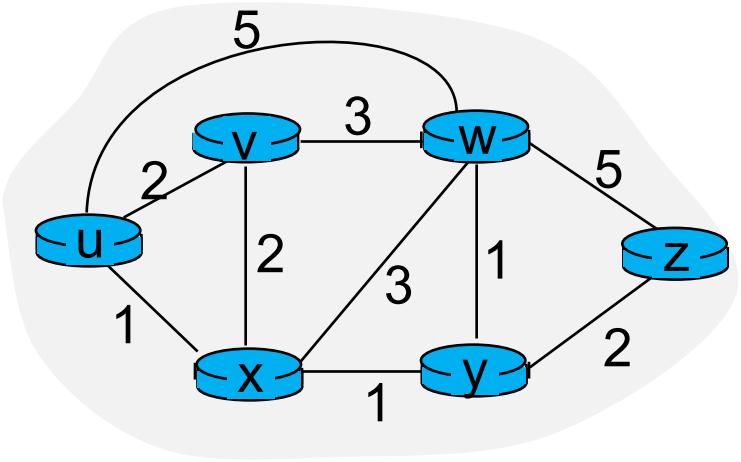
N = set of routers

= { 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) }

Link costs



$c(x_i,x_j)$ = cost of link (x_i,x_j)
$c(w,z) = 5$

What is cost c(x,y)?

Q: how to set cost?

- Always 1
- Related to bandwidth
- Inversely related to congestion
- Actual cost for ISP to use link
- …

Q: What's the least-cost path between u and z?

$c(u,x) + c(x,y) + c(y,z)$

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

Routing algorithm: algorithm that finds least-cost path

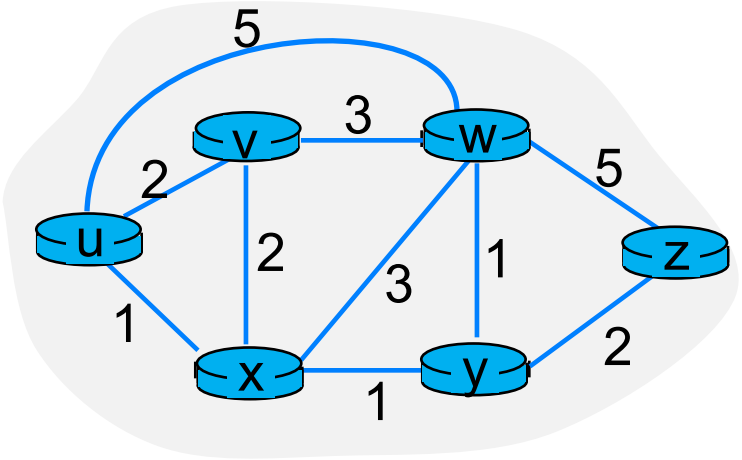
Classifying routing algorithms

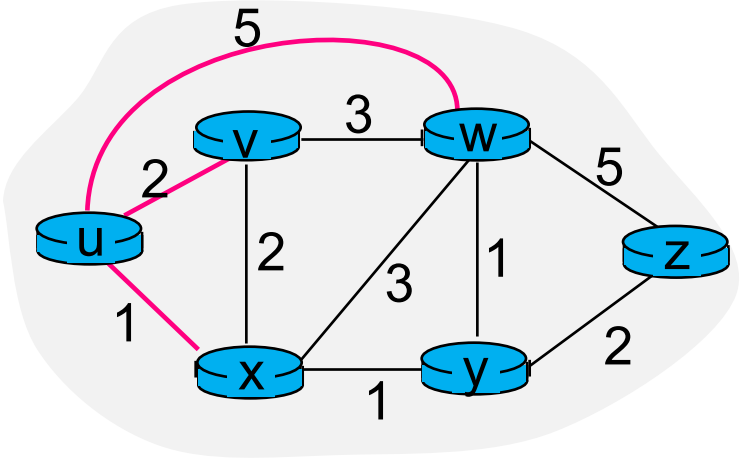
Global information

- global link state algorithms
- all routers have complete topology, link cost info
- *exchange info only about neighbors but with all nodes*

Local/decentralized information

- decentralized distance vector algorithms
- router knows only physically-connected neighbors, link costs to neighbors
- iterative computation
- *exchange info about all nodes but only with neighbors*





Both are used on Internet. First cover abstractly and then talk about specific Internet protocols (OSPF, BGP, RIP, …)

Control Plane

LINK STATE ROUTING

Dijkstra's algorithm

Link state: i.e., network topology, link costs

- known to all nodes, accomplished via link state broadcast
  - msg about a node's neighbors sent to every other node in network
- all nodes have same global info

Computes least cost paths

- from one "source" node to all other nodes
- obtain forwarding table for that node

Given path, put 1st hop router for each dst in forwarding table

Iterative

- after k iterations, know least cost path to k destinations
  - if n nodes, loop n times

Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

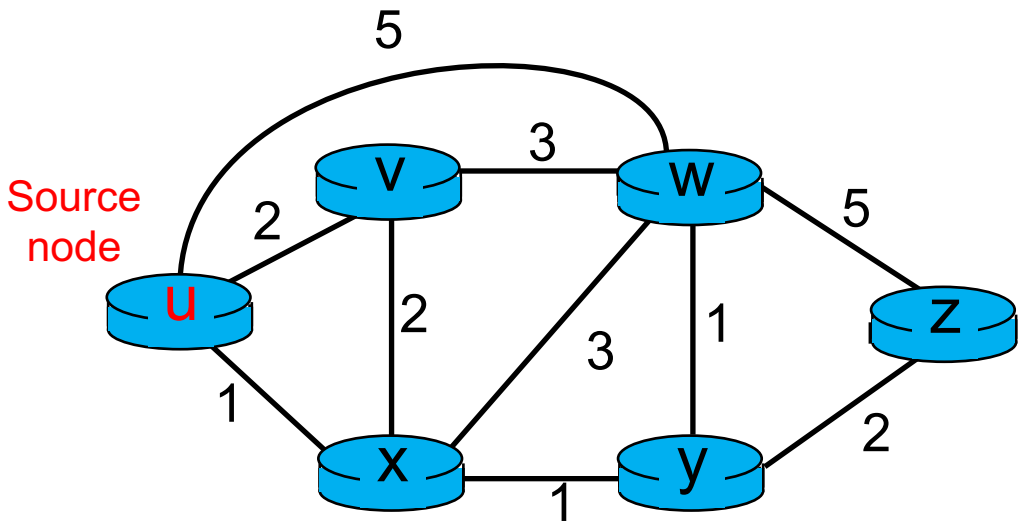
N': set of nodes whose least cost path is **definitively known**

i and j are arbitrary nodes in graph

u will be our starting (aka source) node
k is any arbitrary node

At a give node on path to k, what is node before that node on path?

We don't just know a path to these destinations, we know definitively the least cost path. Essentially building shortest path tree



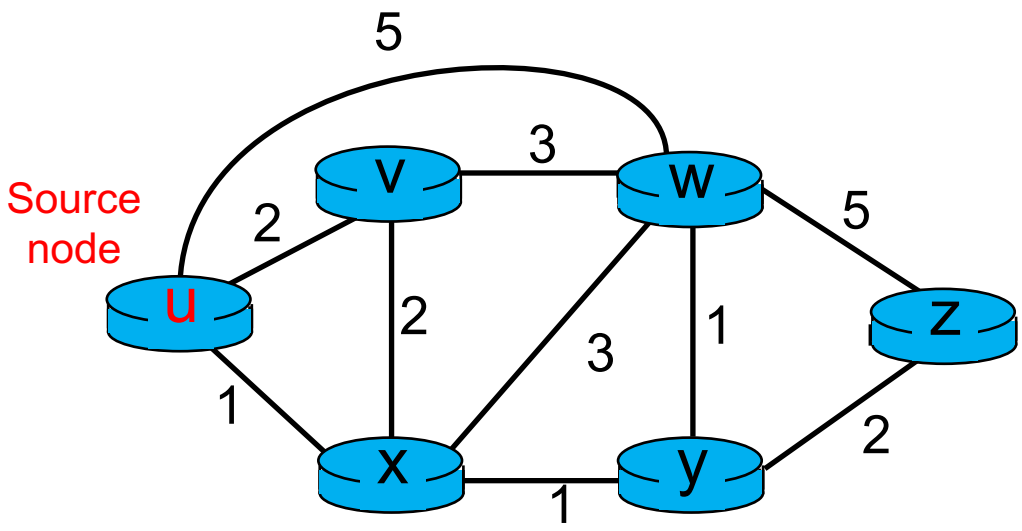
Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**



Initialization

N' = {u}

for all nodes j

if j adjacent to u

then D(j) = c(u,j)

else D(j) = ∞

Dijkstra's algorithm

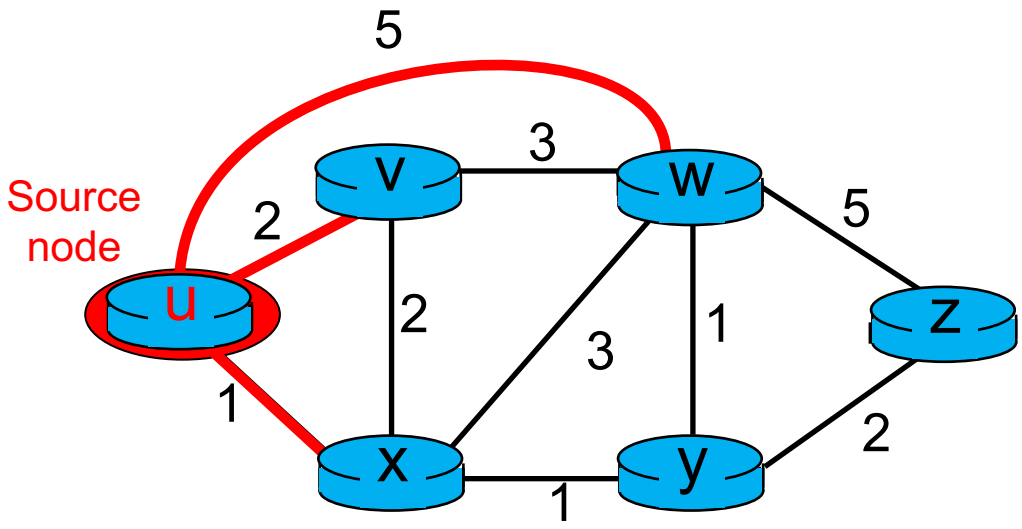
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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



Initialization

N' = {u}

for all nodes j

if j adjacent to u

then D(j) = c(u,j)

else D(j) = ∞

Dijkstra's algorithm

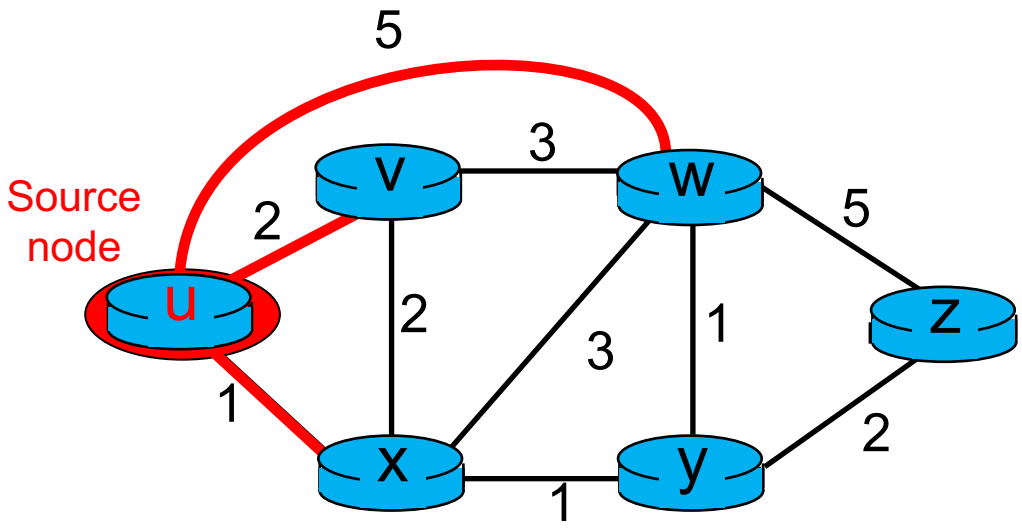
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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						
2						
3						
4						
5						



Initialization

N' = {u}

for all nodes j

if j adjacent to u

then D(j) = c(u,j)

else D(j) = ∞

Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

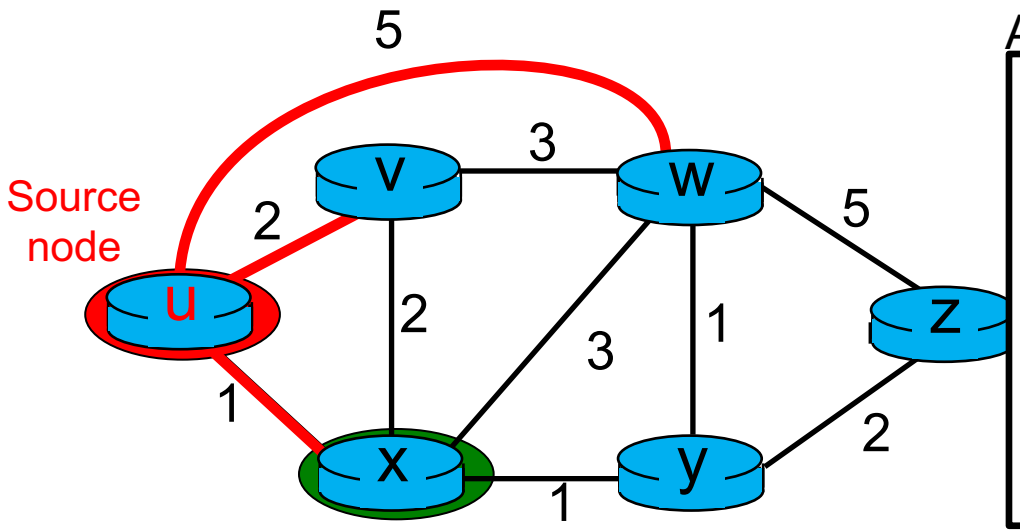
N': set of nodes whose least cost path is **definitively known**

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		x is not in N', and D(x) is lowest				
3						
4						
5						

Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'



Now we know the *lowest cost path from u to x.* Why?

Any other path from u to x must go through *neighbor of u to get to x.* But we just looked at all neighbors of u

Dijkstra's algorithm

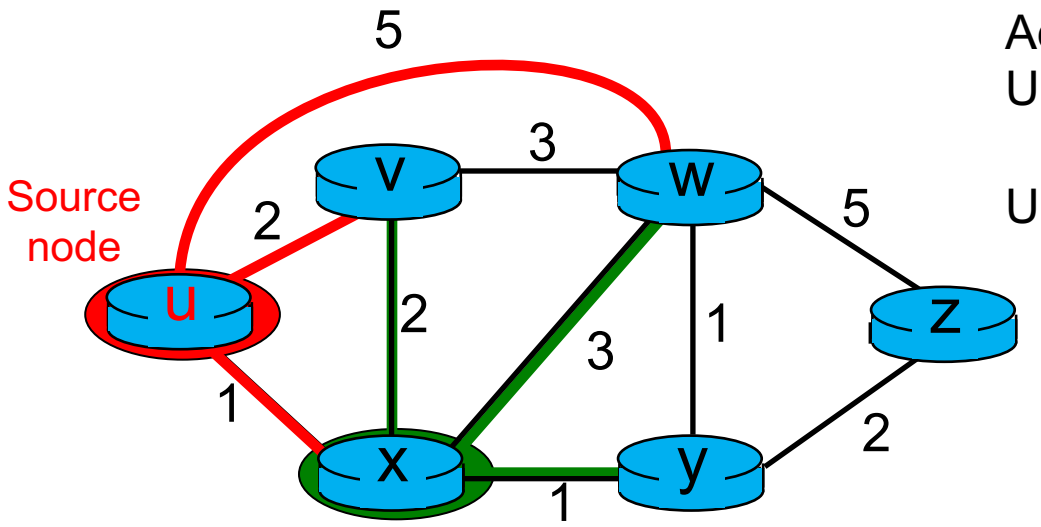
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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						
3						
4						
5						



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

Now we check whether any *neighbors of x that are not in N'* can be reached with lower cost path by *first going through x*

Dijkstra's algorithm

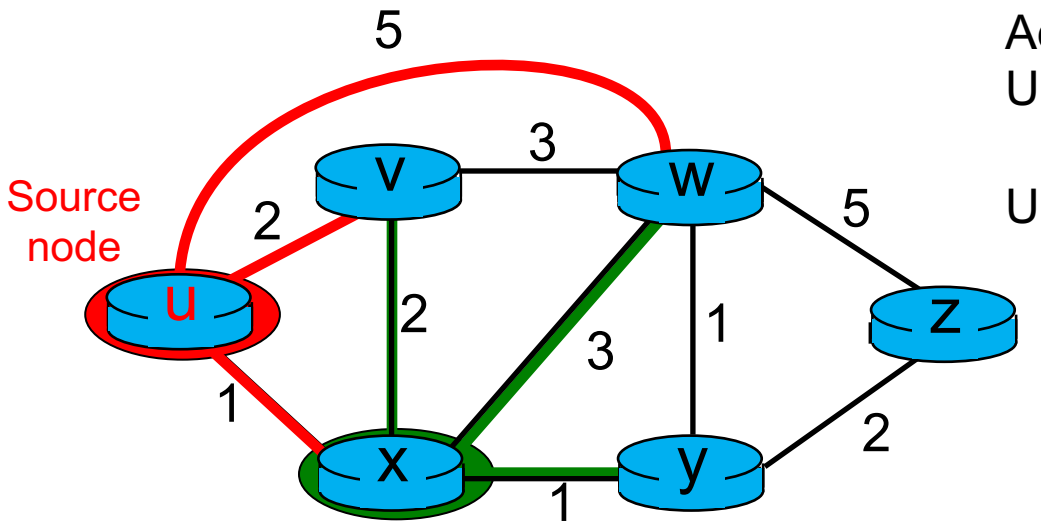
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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				
2		D(v)				
3		= min(D(v), D(x)+c(x,v))				
4		= min(2, 1+2)				
5						



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

3 min: compute the updated values of D(v), D(w), D(y)

Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

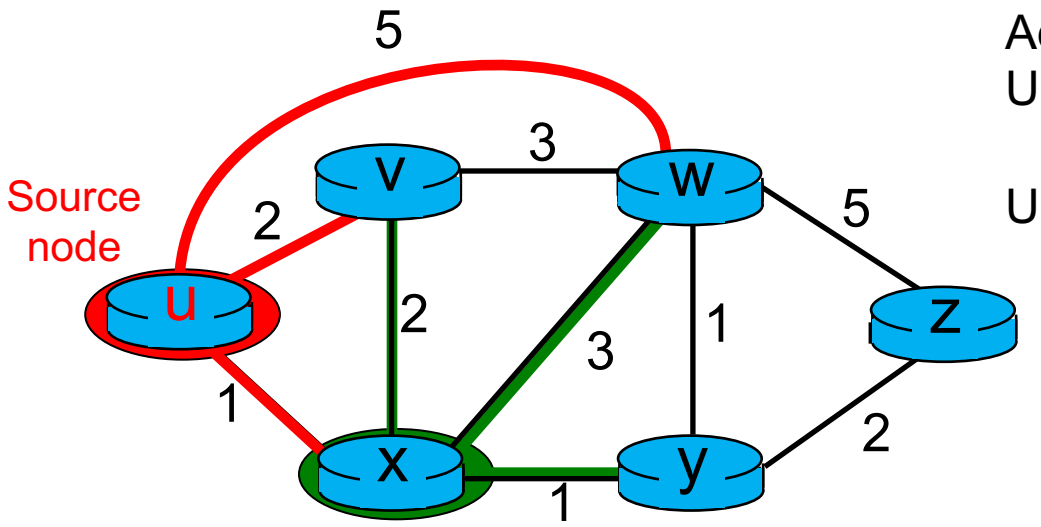
D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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						
3						
4						
5						

D(w)
= min(D(w), D(x)+c(x,w))
= min(5, 1+3)



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

3 min: compute the updated values of D(v), D(w), D(y)

Dijkstra's algorithm

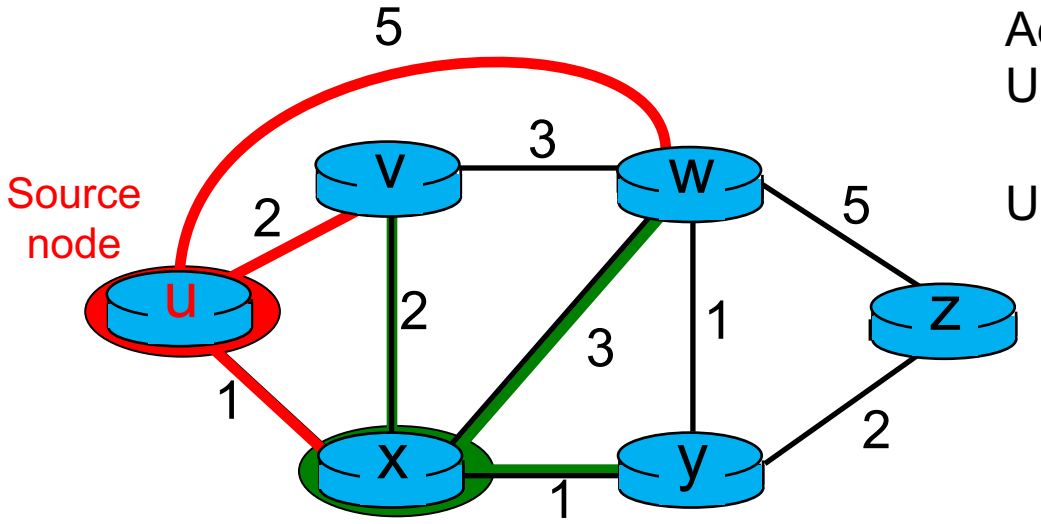
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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 is in N', don't update		
3						
4						
5						



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

3 min: compute the updated values of D(v), D(w), D(y)

Dijkstra's algorithm

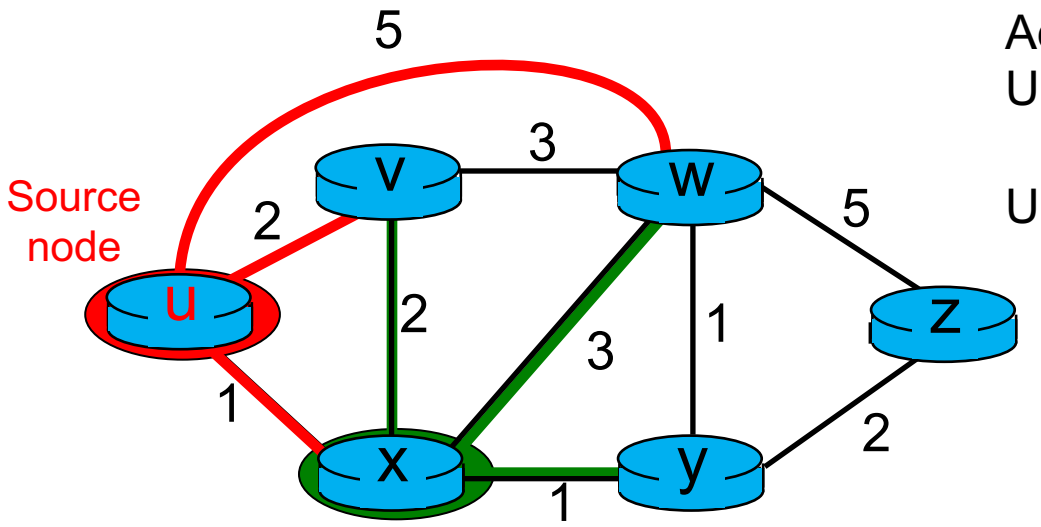
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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					D(y)	
3					= min(D(y), D(x)+c(x,y))	
4					= min(∞, 1+1)	
5						



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

3 min: compute the updated values of D(v), D(w), D(y)

Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

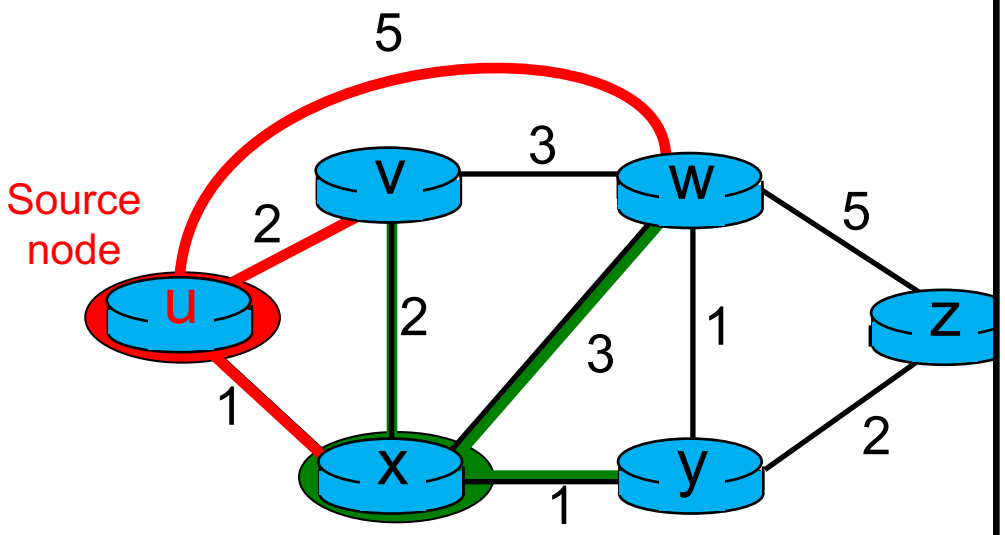
D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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						
3						
4						
5						

D(z): z is not a neighbor of x so don't update



Now we know the *lowest cost path from u to y.* Why?

Any other path from u to y must go through *neighbor of u but x is lowest cost neighbor.*

And adding on cost from x to y still gives *lower (same) cost than even to just go to other neighbors of u.*

Dijkstra's algorithm

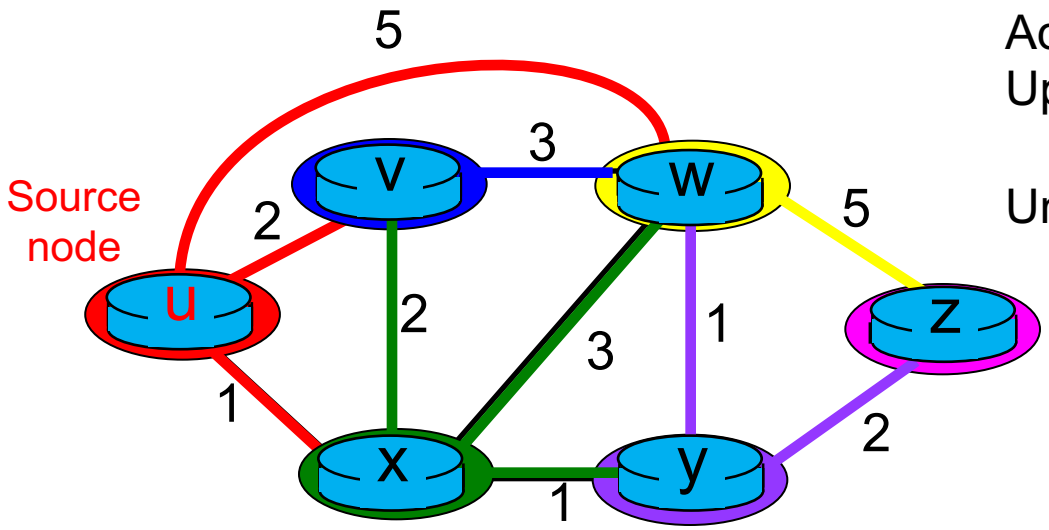
c(i,j): **link cost** from node i to node j

D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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					



Loop

Find j ∉ N' s.t. D(j) is min

Add j to N'

Update D(k) for all neighbors k ∉ N' of j

D(k) = min(D(k), D(j)+c(j,k))

Until all nodes in N'

Dijkstra's algorithm

c(i,j): **link cost** from node i to node j

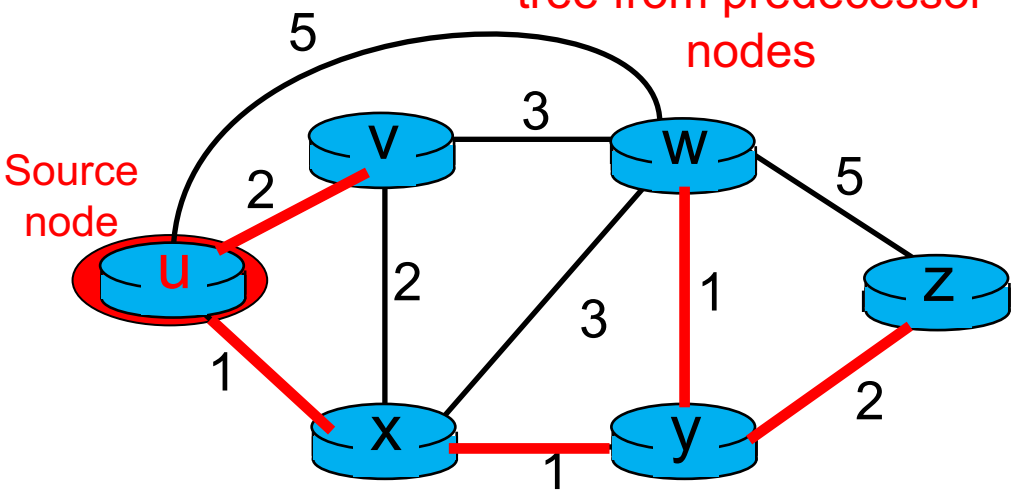
D(k): **current cost** from source u to destination node k

p(k): **predecessor node** along path from source u to k

N': set of nodes whose least cost path is **definitively known**

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					

1. Build shortest path tree from predecessor nodes



2. Build forwarding table at u

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

Algorithm complexity with n nodes

Each iteration: need to check all nodes not in N'

- n in 1st iteration, n-1 in 2nd iteration, n-2 in 3rd iteration …
- n(n+1)/2 comparisons: $O(n^2)$, more efficient implementations possible

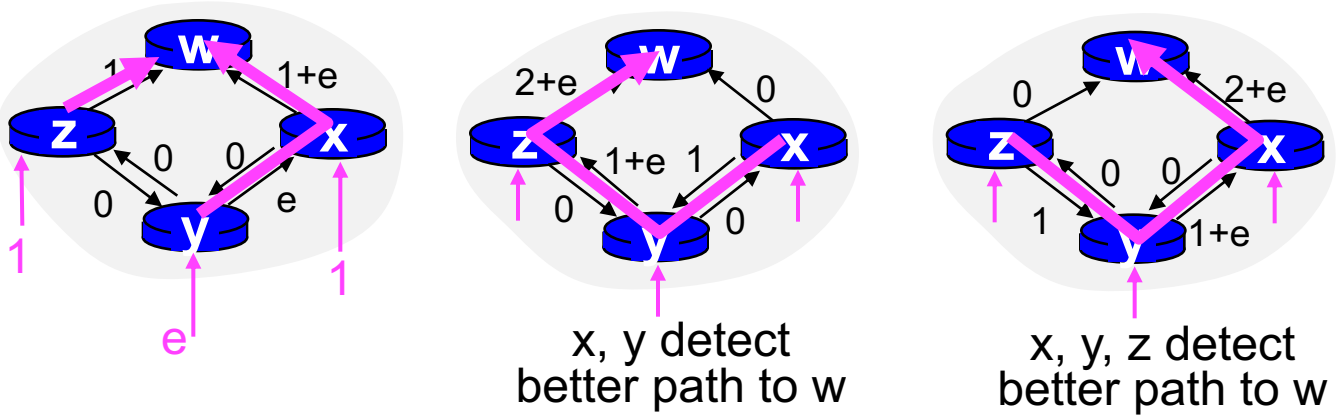
Network is dynamic

- link goes down: link state broadcast
- router goes down: remove link and all nodes recompute

Oscillations possible

- when congestion or delay-based link cost

initially … recompute routing … recompute



x, y detect better path to w

x, y, z detect better path to w

Need to prevent routers from synchronizing computations:
Have routers randomize when they send out link advertisements