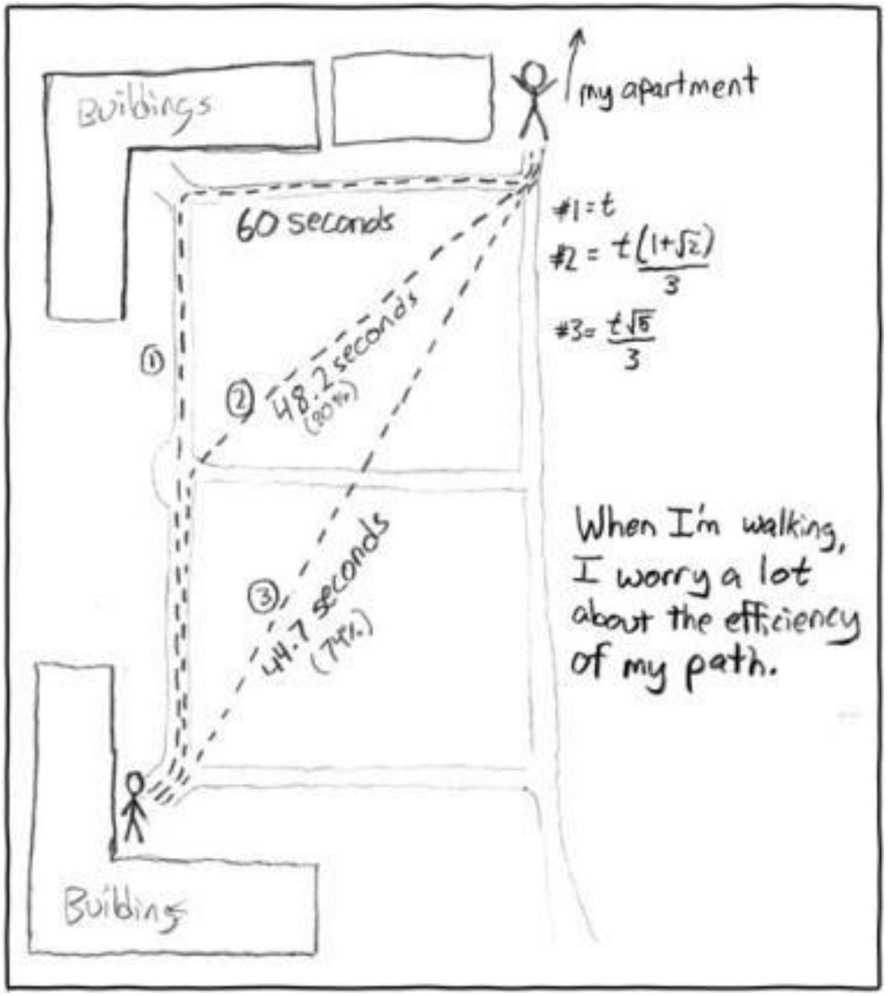
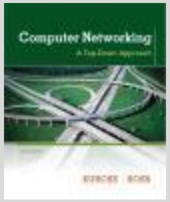
Intra-AS Routing



http://xkcd.com/85/

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley



Chapter 4: outline

4.1 Introduction

4.2 Virtual circuit and datagram networks

4.3 What's inside a router

4.4 IP: Internet Protocol

- Datagram format
- IPv4 addressing
- Network Address Translation (NAT)
- DHCP
- ICMP
- IPv6
- IPsec

4.5 Routing algorithms

- Distance vector
- Link state
- Hierarchical routing

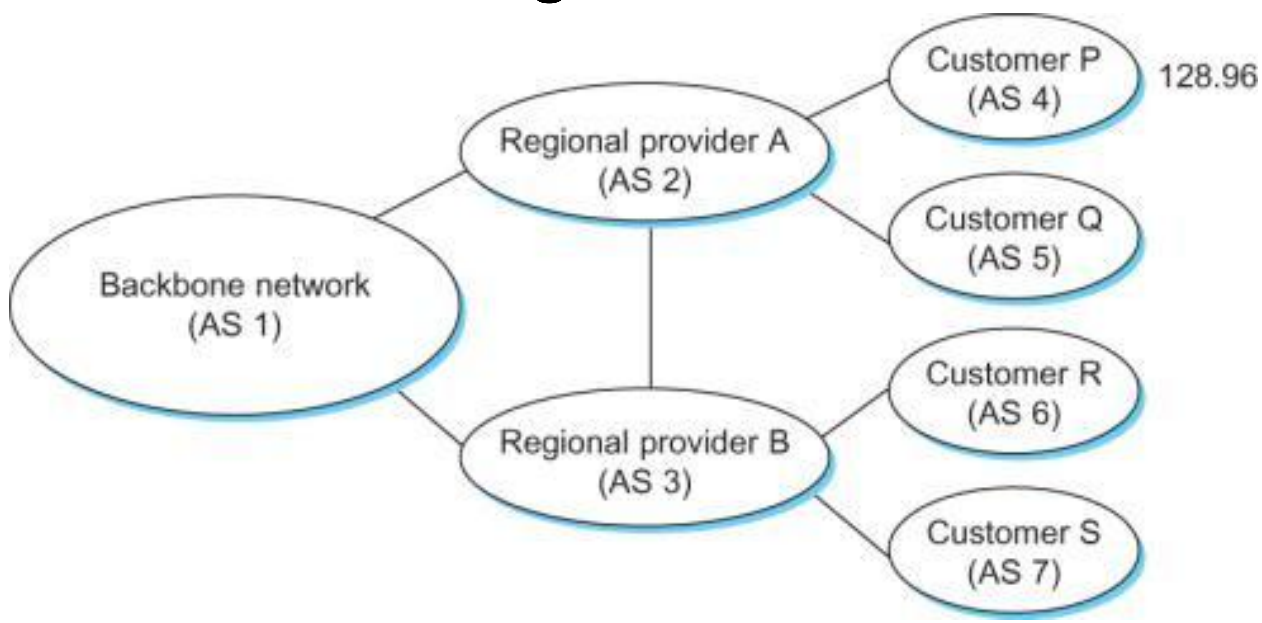
4.6 Routing in the Internet

- RIP
- OSPF
- BGP

4.7 Broadcast and multicast routing

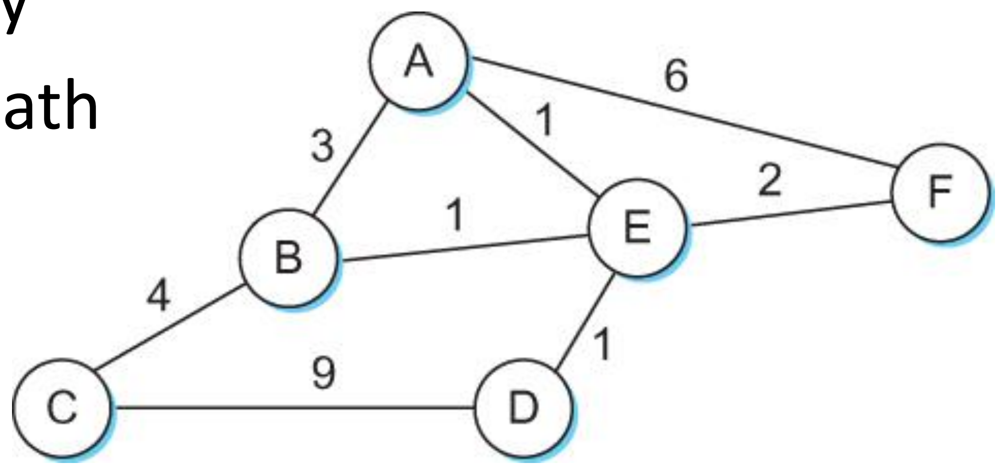
Autonomous System (AS)

- Autonomous System (AS)
  - Distinct region of admin control
  - Routers/links managed by a single institution
  - Each AS can decide how to route within their AS
  - Today: Intra-AS routing
  - Next time: Inter-AS routing



Network as a graph

- Nodes:
  - Hosts, switches, routers, networks
- Edges:
  - Network links
  - May have an associated cost
- Basic problems:
  - Learning the topology
  - Finding lowest cost path



Routing protocols

- Distributed algorithm
  - Running on many devices
  - No central authority
  - Must deal with changing topology
- Classes of routing algorithms:
  - Distance vector routing
  - Link state routing
  - Path vector (hierarchical) routing

Distance vector routing

- Each node maintains state
  - Cost of direct link to each of your neighbors
  - Least cost route known to all destinations
- Routers send periodic updates
  - Send neighbor your array
  - When you receive an update from your neighbor
    - Update array entries if new info provides shorter route
  - Converges quickly (if no topology changes)

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

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

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

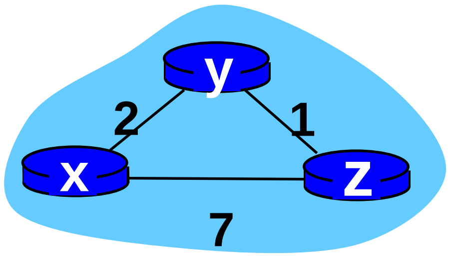
node y table

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



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

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

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

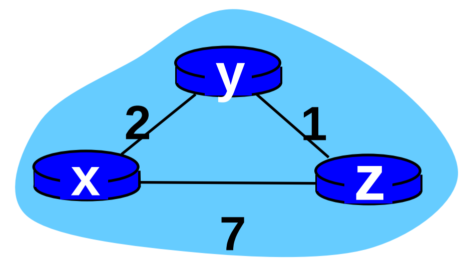
node z table

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 updates

- Periodic updates
  - Automatically send update every so often
  - Lets other nodes know you are alive
- Triggered updates

Wait for (change in local link cost or update from neighbor)

↓

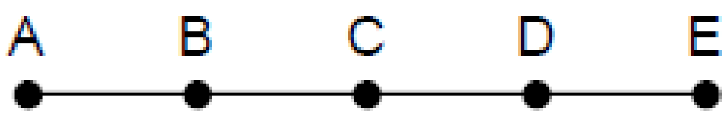
Recompute estimates

↓

If distance to any destination has changed, *notify* neighbors

Link cost change

- What if link added or cost reduced?
  - Update propagates from point of change
  - Network with longest path of N hops:
    - N exchanges, everyone knows of new/improved link
  - "Good news travels fast"



B	C	D	E	
•	•	•	•	Initially
1	•	•	•	After 1 exchange
1	2	•	•	After 2 exchanges
1	2	3	•	After 3 exchanges
1	2	3	4	After 4 exchanges

Link cost change

- What if link deleted or cost increased?
  - Problem: Neighbor has a path somewhere, but you don't know if it goes through you
- Count to infinity problem
  - "Bad news travels slow"

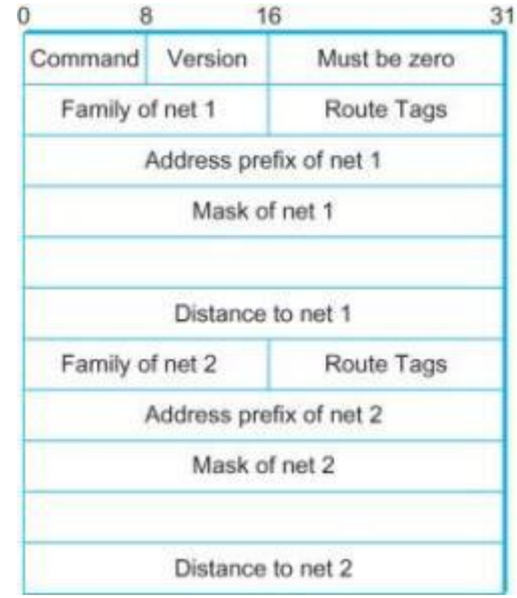
A	B	C	D	E	
•	•	•	•	•	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	•	•	•	•	

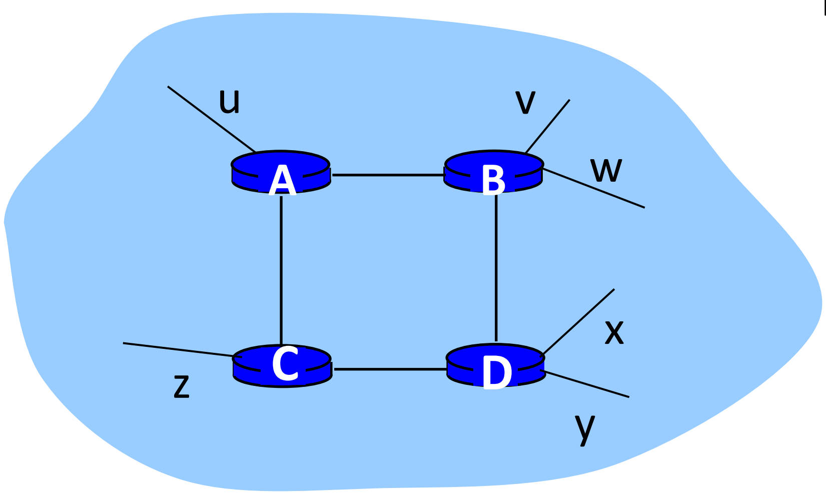
Count-to-infinity

- Various ways to "fix":
  - Use a small values for infinity, e.g. 16
    - Limits network size to 15 hops
  - Split horizon with poisoned reverse
    - Track where you learned the route
    - e.g. (E, 2, A), I learned a cost 2 route to E from A
    - When B updates A, sends (E, ∞)
    - Only works for two node routing loops
  - Holddown timer
    - Start a timer when a network becomes unreachable
    - Don't update until timer expires

RIP

- Routing Information Protocol (RIP)
  - Distance-vector protocol
  - Used in original ARPANET, in BSD
  - All links costs 1
  - Advertise every 30 seconds
  - Small networks, < 16 hops
  - Runs over UDP

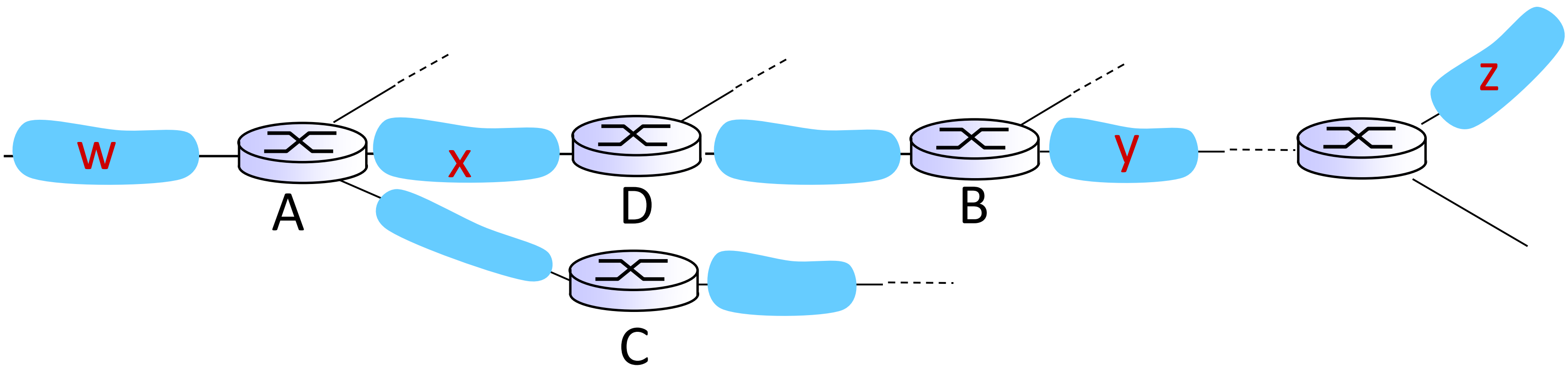




From router A to destination *subnets:*

subnet	hops
u	1
v	2
w	2
x	3
y	3
z	2

RIP: example



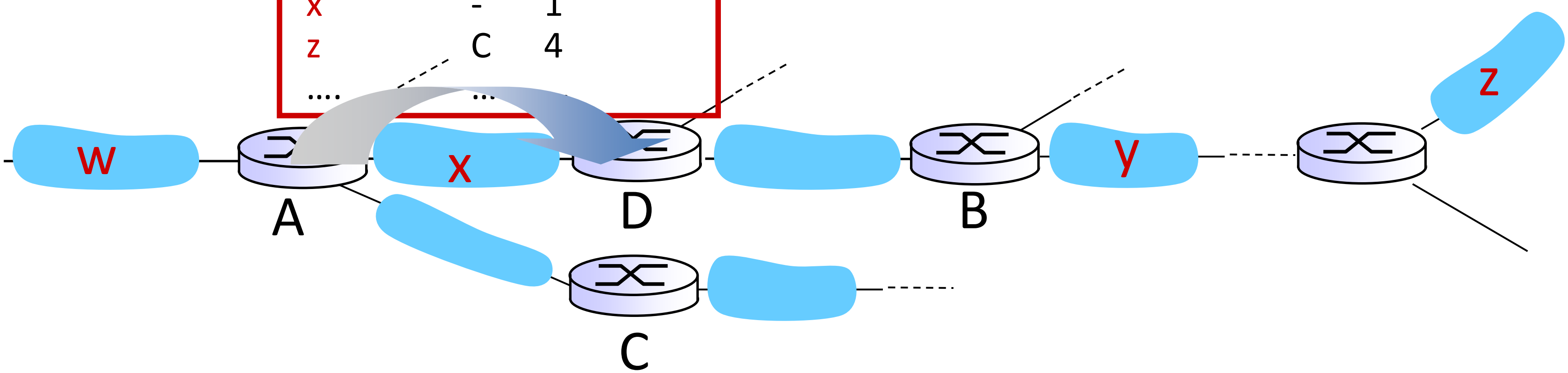
routing table in router D

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	B	7
x	--	1
….	….	….

RIP: example

A-to-D advertisement

dest	next	hops
w	-	1
x	-	1
z	C	4
….	…	



routing table in router D

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	~~B~~ A	~~7~~ 5
x	--	1
….	….	….

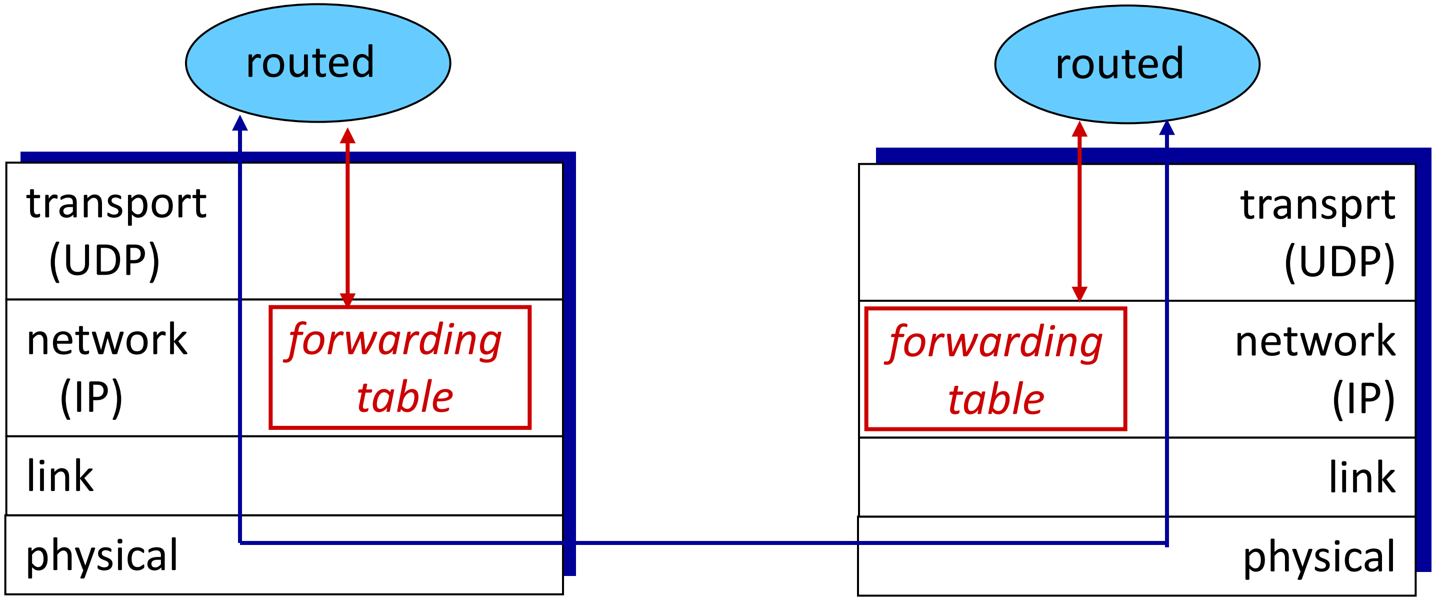
RIP: link failure, recovery

If no advertisement heard after 180 sec --> neighbor/link declared dead

- Routes via neighbor invalidated
- New advertisements sent to neighbors
- Neighbors in turn send out new advertisements (if tables changed)
- Link failure info quickly (?) propagates to entire net
- *Poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)

RIP table processing

- RIP routing tables managed by *application-level* process called route-d (daemon)
- Advertisements sent in UDP packets, periodically repeated

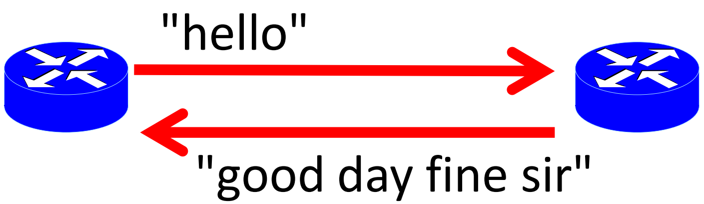


Link state routing

- Link state routing
  - Each router tracks its immediate links
    - Whether up or down
    - Cost of link
  - Each router broadcasts link state
    - Information disseminated to all nodes
    - Routers have global state from which to compute path
  - e.g. Open Shortest Path First (OSPF)

1. Learning about your neighbors

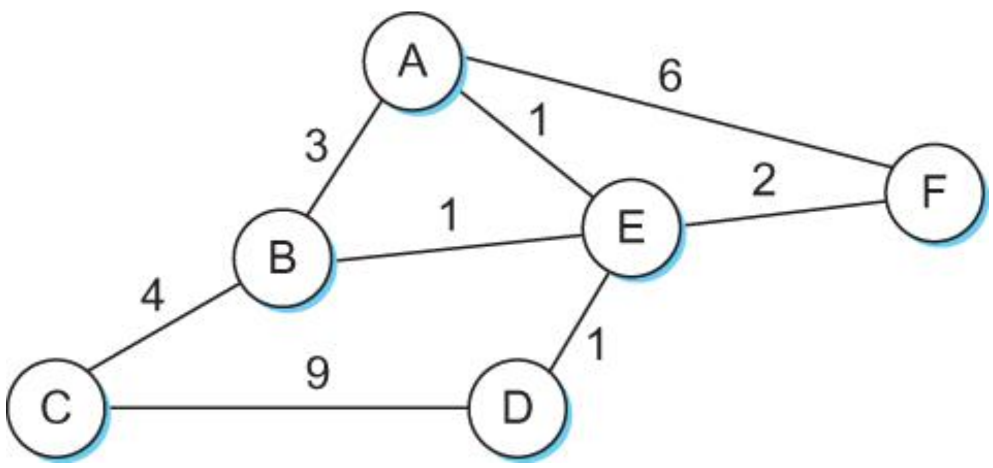
- Beaconing
  - Find out about your neighbors when you boot
  - Send periodic "hello" messages to each other
  - Detect a failure after several missed "hellos"



- Beacon frequency is tradeoff:
  - Detection speed
  - Bandwidth and CPU overhead
  - Likelihood of false detection

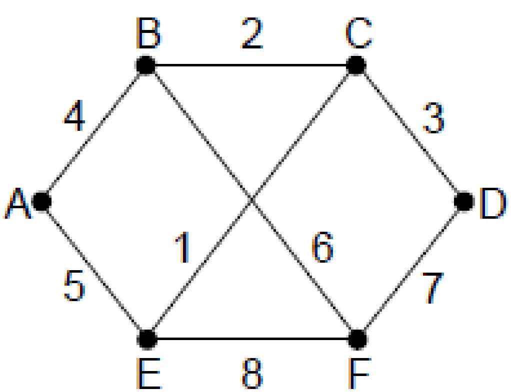
2. Setting link costs

- Assign a link cost for each outbound link
  - Manual configuration
  - Automatic
    - Inverse of link bandwidth
      - 1-Gbps cost 1
      - 100-Mbps cost 10
    - Measure latency by sending an ECHO packet



3. Building link state packets

- Package info into a Link State Packet (LSP)
  - Identity of sender
  - List of neighbors
  - Sequence number of packet
  - Age of packet

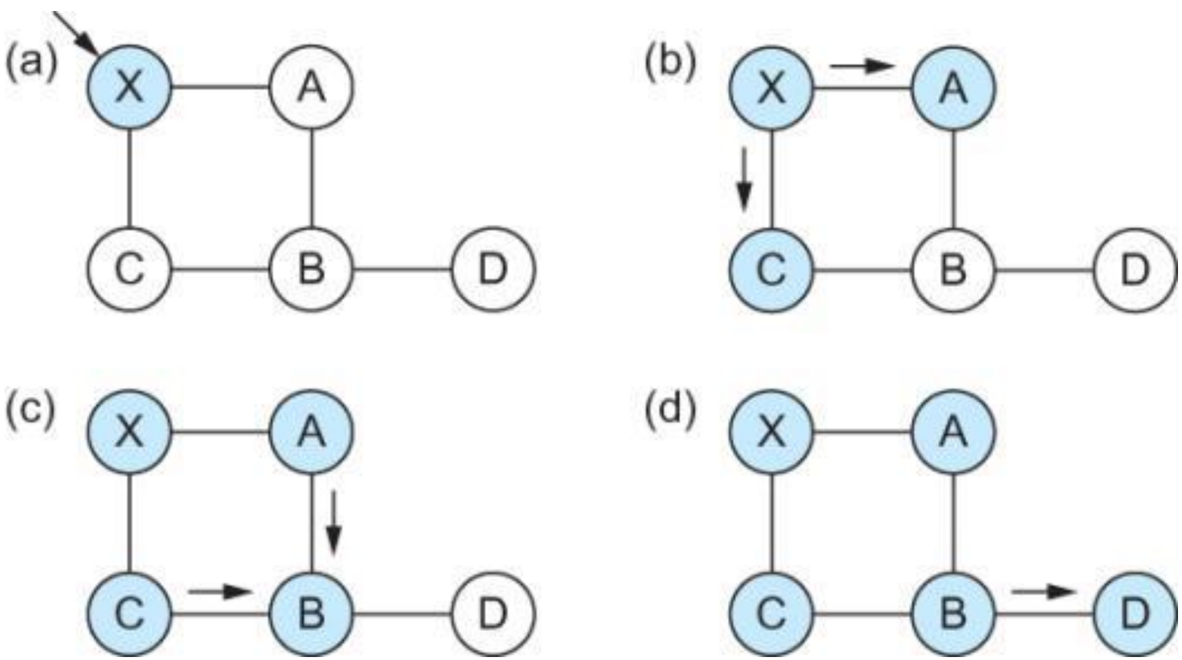


Link State Packets

A		B		C		D		E		F	
Seq.		Seq.		Seq.		Seq.		Seq.		Seq.	
Age		Age		Age		Age		Age		Age	
B	4	A	4	B	2	C	3	A	5	B	6
E	5	C	2	D	3	F	7	C	1	D	7
		F	6	E	1			F	8	E	8

4. Distributing link state

- Flooding
  - Send your LSP out on all links
  - Next node sends LSP onward using its links
    - Except for link it arrived on



a) LSP arrives at node X
b) X floods LSP to A and C
c) A and C flood LSP to B (but not X)
d) flooding complete

4. Distributing link state

- Making flooding reliable
  - Use acknowledgments and retransmissions between routers
  - Use sequence numbers
    - Discard info from packets older than your current info
  - Time-to-live TTL keeps LSP from being endlessly forwarded
- When to distribute?
  - Periodic timer
  - On detected change

5. Computing routes

- Router has accumulated full set of LSPs
  - Construct entire network graph
  - Shortest path from A to B?

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's algorithm

- Net topology known to all nodes
  - All nodes have same info
- Computes least cost paths from some source node to all other nodes
  - Gives *forwarding table* for that node
- Iterative: after k iterations, know least cost path to k dest's

Dijkstra's algorithm

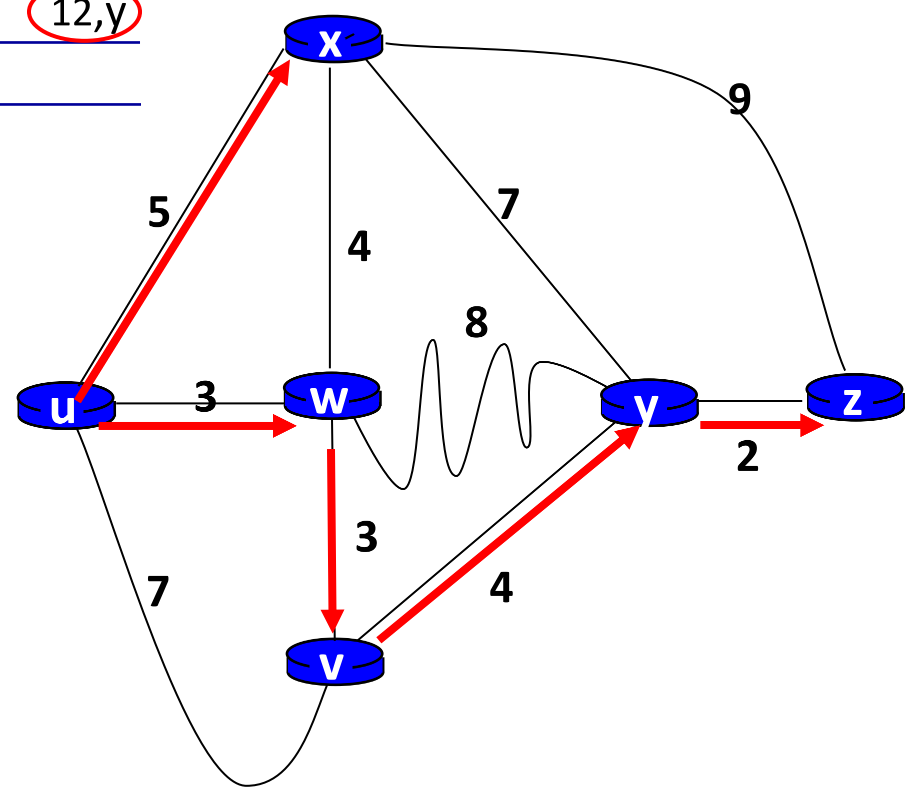
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
       then D(v) = c(u,v)
    else D(v) = ∞

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'
```

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					

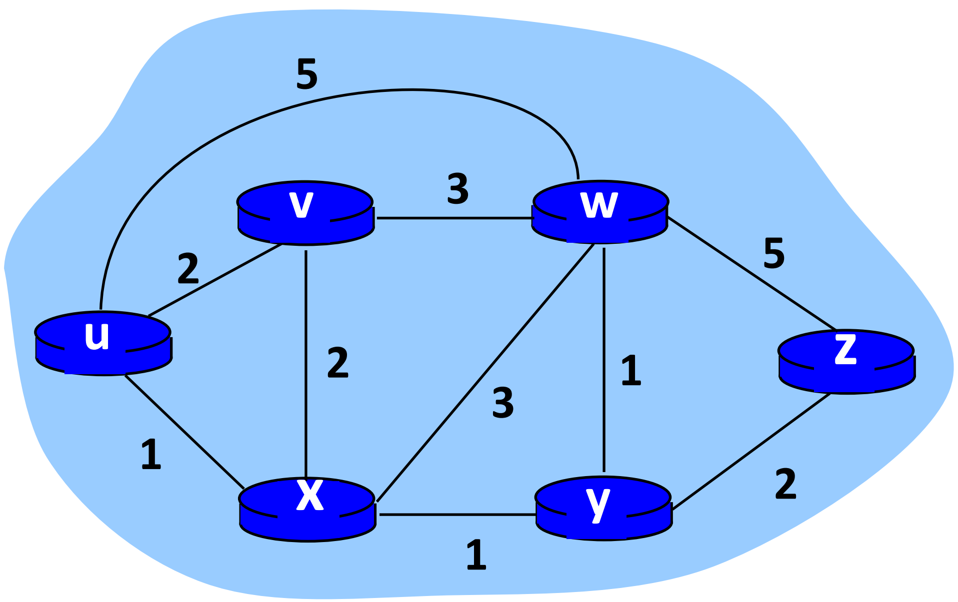
Notes:

- Construct shortest path tree by tracing predecessor nodes
- Ties can exist (can be broken arbitrarily)



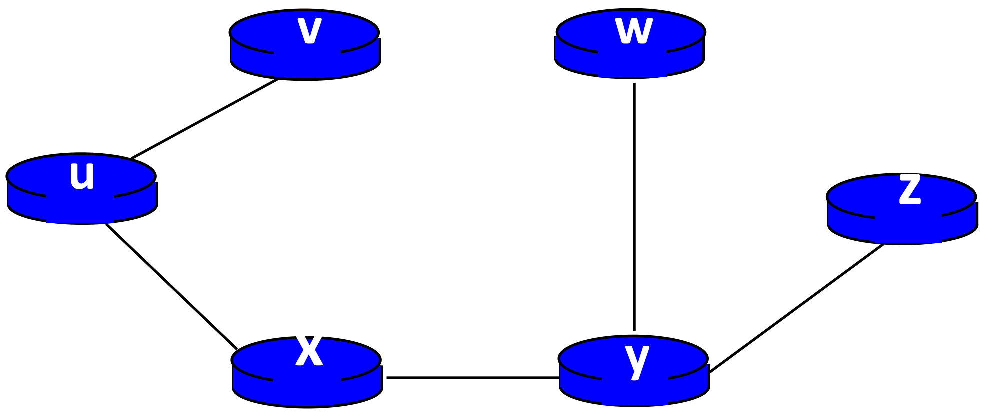
Dijkstra's algorithm: another example

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					



Dijkstra's algorithm: another example

Resulting shortest-path tree from u:



Resulting forwarding table in u:

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

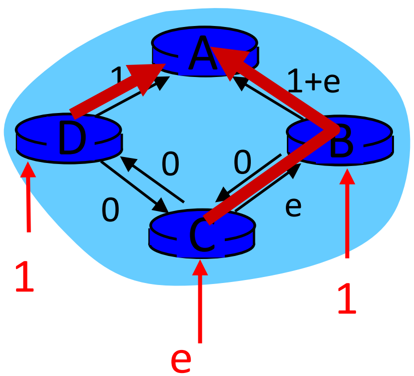
Dijkstra's algorithm: discussion

Algorithm complexity: n nodes

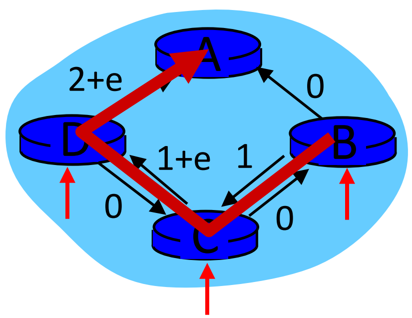
- Each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- More efficient implementations possible: O(nlogn)

Oscillations possible:

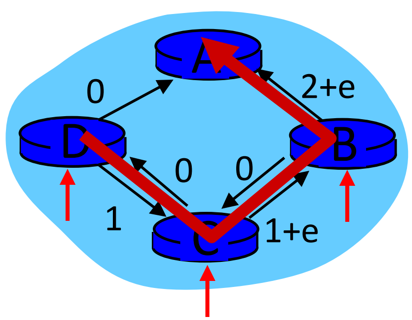
- e.g. support link cost equals amount of carried traffic:



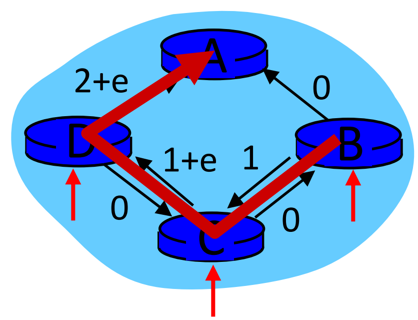
Initially



Given these costs, find new routing…. Resulting in new costs



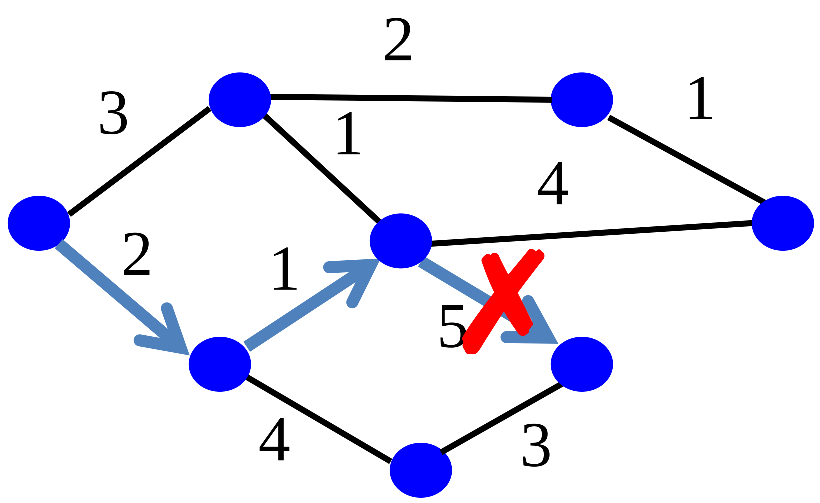
Given these costs, find new routing…. Resulting in new costs



Given these costs, find new routing…. Resulting in new costs

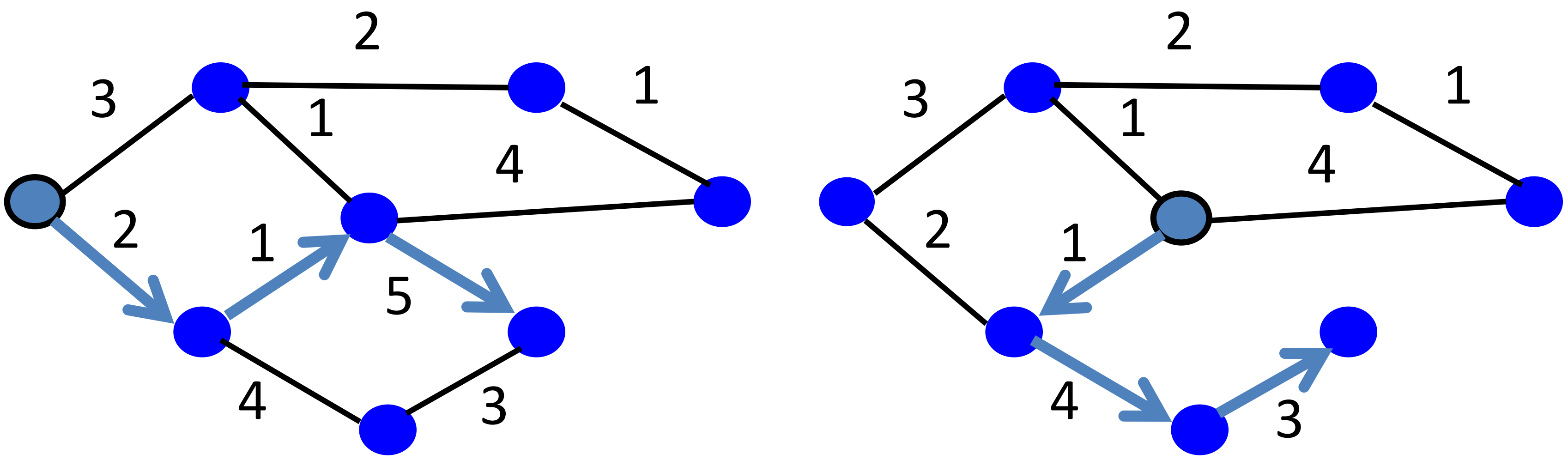
Transient disruptions

- Detection delay
  - Failures are not detected immediately
  - Router may forward packet into a "blackhole"
  - Chance depends on frequency of "hello" messages



Transient disruptions

- Inconsistent link-state
  - Some routers know about a failure, others don't
  - Shortest path no longer consistent
  - Can causes transient forwarding loops



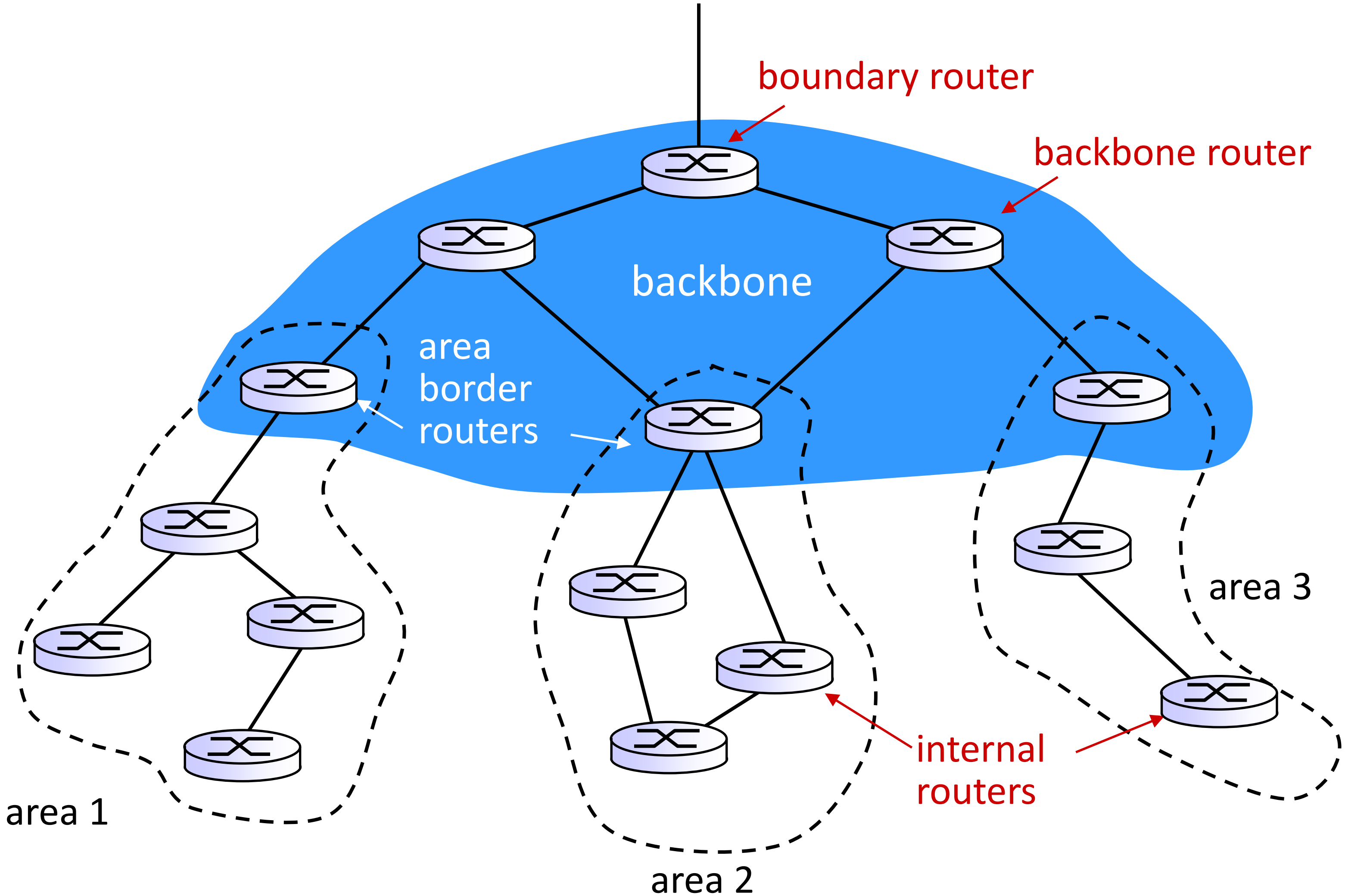
OSPF (Open Shortest Path First)

- Open: publicly available
- Uses link state algorithm
  - LS packet dissemination
  - Topology map at each node
  - Route computation: Dijkstra's algorithm
- OSPF advertisements:
  - One entry per neighbor
  - Flooded to entire AS
  - Carried in OSPF messages directly over IP
- IS-IS routing protocol
  - Nearly identical to OSPF

OSPF advanced features (not in RIP)

- Security: All OSPF messages authenticated
  - To prevent malicious attacks
- Multiple same-cost paths allowed
  - Only one path allowed in RIP
- Multiple link cost metrics for different TOS
  - e.g. satellite link cost set low for best effort ToS; high for real time ToS
- Integrated uni- and multicast support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- Hierarchical OSPF in large domains

Hierarchical OSPF



Hierarchical OSPF

- *Two-level hierarchy:* local area, backbone
  - Link-state advertisements only in area
  - Each node has detailed area topology; only knows direction (shortest path) to nets in other areas
- *Area border routers:*
  - Summarize distances to nets in own area, advertise to other Area Border routers
- *Backbone routers:*
  - Run OSPF routing limited to backbone
- *Boundary routers:*
  - Connect to other AS's

Convergence delay

- Sources of delay:
  - Time to detect failure
  - Time to flood link-state info
  - Shortest path computation
  - Creating the forwarding table
- Before convergence:
  - Lost packets due to blackholes, TTL expiry
  - Looping packets
  - Out of order packets
  - Bad for Voice over IP, gaming, video

Reducing convergence delay

- Detect failures faster
  - Increase beacon frequency
  - Link-layer technologies that can detect failures
- Faster flooding
  - Flood immediately on a change
  - LSP sent with high-priority
- Faster computation
  - Faster processors in routers
  - Faster algorithms
    - e.g. incremental Dijkstra's
  - Faster forwarding table update
    - e.g. data structures supporting incremental updates

Distance vector vs. Link state

Distance vector	**Link state**
Knowledge of neighbors' distance to destinations	Knowledge of every router's links (entire network graph)
Router has O(# neighbors * # nodes)	Router has O(# edges)
Messages only between neighbors	Messages between all nodes
Trust a peer's routing computation	Trust a peer's info Do routing yourself
Bellman-Ford algorithm	Dijkstra's algorithm
<u>**Advantages:**</u> Less info has to be stored Lower computation overhead	<u>**Advantages:**</u> Fast to react to changes

Summary

- Intra-AS routing, two major types:
  - Distance vector
    - Router only know about its neighbors
    - RIP protocol
      - Original protocol on the ARPANET
      - Limited to networks < 16 hops
  - Link state
    - Full state of network known by each router
    - OSPF protocol
      - More advanced features: security, multiple paths, multiple cost metrics, routing areas